2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

RE-VISIONING OF THE AUTOMATIC GRADING/LEARNING SYSTEM

Jason Felds

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2012

Approved by

Advisory Committee

Dr. Bryan Reinicke	Dr. Laurie Patterson
Kevin Matthews	Dr. Thomas Janicki, Chair

Accepted By

Dean, Graduate School

Abstract

Re-visioning of the Automatic Grading/Learning System. Felds, Jason, 2012. Capstone Paper, University of North Carolina Wilmington.

In 2008, the ISOM department at UNCW implemented a system that would allow students in certain classes to be able to submit electronic files to be automatically graded as part of the existing grade book system. While that system has been successful, enhancements are desired to make the system more user-friendly and to grade additional features. A number of flaws have been uncovered with the Automatic Grading/Learning System (AGLS) that should be corrected. The goal of this project is to make it easier for the professor to create and grade assignments, but also to make the AGLS more generic so it may be used by other grade book systems.

Table of Contents

Page

Chapter 1: Introduction

As more students go to a college or university, class sizes often get larger. When this happens, the assignments that professors give to the students can take longer to grade due to that increase in size (Matthews, Development of an Adaptive Grading/Learning System (AGLS), 2008, p. 8). Getting more professors or assistants to grade the assignment to speed up the process can be problematic: not only do the other professors and assistants need to grade their own work, but it has been stated that "to preserve consistency…when grading…it is best for a single individual to grade every student's response to a given question." (Kay D. , 1998, p. 133) The AGLS was designed to allow for quicker and easier grading of specific electronic homework assignments so that students would get more feedback on their assignments.

It should be noted, however, that there are a few flaws with the present AGLS that are not easy to fix in its current implementation. The AGLS was built using practices that are not up to the currently accepted 'best practices' as of this time. Today's best practices would indicate a need for better source control, testing, and code isolation within this system. The AGLS is also embedded inside Entropy, an internal grade book program. This renders the AGLS non-modular, or non-generic, in this state. Furthermore, while the AGLS did solve the goal of making it easier for students to get their grades faster, enhancements could be made so that professors can more easily create and submit answer keys for assignments.

The goal of this capstone is to start the rebuilding of the AGLS into a more modular, marketable platform that can be used by other schools and universities. The complete rebuilding process will take a substantial amount of time; as such, only select

features and tasks will be focused on for this capstone. Any goals not focused on for the upcoming semester can be viewed as future research. However, any code that is written should be clear and modular enough so that if anyone wishes to make improvements, it will be easy to do so.

Professors can potentially teach a large number of students each semester: over 70 students taught across all classes is not uncommon. Most professors that use the AGLS in MIS 213, Intro to Information Systems & Technologies, assign at least six homework assignments to the students, roughly split in half between Microsoft Excel and Microsoft Access. This means that a minimum of 420 assignments among all students are graded each semester. AGLS was designed to work in such a capacity, and more.

A number of enhancements have been requested by the professors for AGLS over the years of its existence to make the system even more powerful and robust than it currently is. These requests range from adding more features of Access and Excel to grade to more options for creating answer keys, and from making the AGLS accept more than just Microsoft Office files to grade to making the system marketable and available for other colleges to use in the future. It is through this project that as many of these requests as possible will be acknowledged and enabled.

Chapter 2: Review and Analysis of Literature

In order to build a new grading system, research is required. This means looking into a variety of articles and studying how the predecessor of this system behaved. Some definitions are also provided.

Definition of Terms

Due to the nature of this project, a number of different words, phrases, and abbreviations will be used throughout this document. The terms are listed below in alphabetical order.

.NET Framework. The .NET Framework (pronounced "dot net framework") is a framework provided by Microsoft to allow different languages to interact with each other as long as the language follows the framework's guidelines. The most common languages used by this framework for web development purposes are C# and VB.

AGLS. AGLS is the Automatic Grading Learning System. This project is meant to be a second version of the original.

ASP.NET. ASP.NET is the web framework Microsoft made to allow web sites and services be created and hosted. Its name and origin comes from ASP, or Active Server Pages, which is the original framework that was used prior to the .NET Framework being around (Microsoft, 2001).

Binary. A binary is a program that can be run by a user. This word is interchangeable with executable.

C#. C#, pronounced C Sharp, is a .NET Framework language created by Microsoft meant to be easy to understand and write for a variety of purposes such as

embedded systems and internationalization (ECMA International, 2006, p. 21). This project will primarily use the C# language.

CDN. CDN stands for Content Delivery Network. CDNs are primarily used to serve static content such as JavaScript and CSS files. They work by having multiple servers spread across the world; when a user needs a file on the CDN, the closest server is generally chosen (Yahoo! Inc, 2012).

Creative Commons. Create Commons, sometimes abbreviated as CC, is a form of digital license that attempts to make it easy to share a variety of "creative works" (Creative Commons). There are a number of variations with these licenses, including attribution and commercial clauses.

CSS. CSS stands for Cascading Style Sheets. It is designed to give style to structured documents such as HTML and XML (W3C, 2011). The latest version that all modern browsers should support is CSS 2.1. However, all of these browsers have started to implement various features of the next version of CSS, CSS 3.

Dependency Injection. Dependency Injection, sometimes abbreviated DI or called Inversion of Control (IoC), means that a separate object or assembly will supply an appropriate object implementation for an interface when such an implementation is requested (Fowler, 2004). This basically means implementations can be chosen at runtime rather than compiler time. This methodology can be useful for unit testing, especially when paired with mock objects.

GNU. GNU recursively stands for GNU's Not Linux. It is Unix, or rather, Unix-like. The idea behind GNU is that the operating system and computer programs should be free. More specifically, they prefer thinking of free as in speech, rather than beer. To the

GN crew, “Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software.” (GNU, 1996)

GPL. GPL is short for the GNU General Public License. The license has gone through three major iterations, with the latest being June 2007 at version 3. The key part about the GPL is that if any GPL software is used and/or linked with one’s program, then said program needs to also be licensed under the GPL (Free Software Foundation, Inc., 2007). Due to reasons that will be apparent in the future, relying on GPL code and libraries needs to be avoided whenever possible. This is perhaps one of the most restricted licenses for software that is free.

HTML. HTML stands for Hypertext Markup Language. This is the primary language used to display content on the internet (W3C, 1999). It is composed of various tags surrounded by <angle brackets> and optional attributes inside the angle brackets to dictate how the data is represented. While the structure is very similar to XML, it is not a requirement to have HTML be perfectly formed. It should be noted that HTML 4.01 is what all modern browsers should support, but many of the browsers are supporting newer features of the upcoming HTML 5 specification.

IDE. IDE stands for Integrated Development Environment. It is often a program that consists of at least one code editor and a compiler to turn the code into either an executable or a library to be used with other executables.

Intellisense. Intellisense is a feature of Visual Studio that allows developers to be given a list of suggestions for how to interact with the current code. Thus, Intellisense also allows develops to learn about the code being developed (Microsoft, 2010).

LINQ. LINQ stands for Language-Integrated Query. It allows developers to use a SQL like syntax to retrieve data from a variety of structures instead of, or in addition to, the traditional method calls (Microsoft, 2010).

LGPL. LGPL is short for the GNU Lesser General Public License. While the terms and acronym look similar to the GPL, there is one key difference. If any LGPL programs are linked into one's software, then the software in question can still be licensed under whatever the author wants (Free Software Foundation, Inc., 2007). This means, among other things, that anything that links to LGPL libraries can be sold commercially with little issue.

MSCSIS. MSCSIS stands for the Masters of Computer Science and Information Systems. This is the department that allowed work to be done on the original AGLS.

MVC. MVC is a programming paradigm often referred to as Model-View-Controller. The idea is to use an object-oriented design on the code that is written so that each code file has a distinct purpose and interacts with the other parts only when required. Models often represent the underlying data such as a database table. Views are rendered to the client upon retrieving the data. Controllers call upon the models to get the data and then pass it on to the view (Burbeck, 1992).

NuGet. NuGet is a package manager that can be installed into Visual Studio. It is designed to make it easy for developers to locate and install various third party packages such as templates, libraries, and more (Outercurve Foundation, 2012). The packages have a variety of licensing terms: the packages ones used in this project are designed to be compatible with the written code for this phase. Future iterations beyond this project may involve different packages or licensing terms.

Razor. Razor is a view engine that was introduced with ASP.NET MVC 3. Its goals include allowing C#/VB code in the view, displaying smart information with Intellisense, and being unit testable (Gu, 2010). For reference, older versions of ASP.NET MVC included the Web Forms engine consisting of many .aspx, .ascx, and .master files.

SASS. SASS, sometimes referred to as "Style with Attitude" but properly referred to as "Syntactically Awesome StyleSheets", is an extension of CSS 3 that adds features such as variables, nested styles, and measurement math (File: SASS_REFERENCE, 2012). A separate program will turn the SASS files (either with a .scss or .sass extension) into proper .css files for use on the web.

SQL. SQL stands for Structure Query Language. SQL allows for using a database to gather and join related data from many database tables to put into a result table (IBM, 2006). There are many variants of SQL, such as MySQL, PostgreSQL, SQLite, and SQL Server.

SVG. SVG stands for Scalar Vector Graphics. The format renders graphics through the use of numbers, math, and XML tags; this allows for any image to be rendered cleanly regardless of size or zoom.

TFS. TFS stands for Team Foundation Server. It allows for collaboration between many developers throughout an application's life cycle, and comes with a number of features such as migration tools and version control (Microsoft, 2012). This is an essential tool for working with .NET applications in many work environments.

Visual Studio. Visual Studio, sometimes abbreviated to VS, is the primary IDE used for working with .NET Framework projects on Windows operating systems. The program is available in many editions.

VML. VML stands for Vector Markup Language. Like SVG, it is an XML vector format for rendering graphics cleanly. Microsoft Excel uses VML for some of its images and charts. It should be noted that this format is no longer developed, and Microsoft released a guide on how to transition from VML to SVG (Microsoft, 2010).

Web Forms. Web Forms is the original way to design ASP.NET web sites. It is designed to be similar to developing a Windows form-based application, only it targets the internet instead of just those on a Windows operating system. As Windows applications generally have a concept of state, and the internet usually operates in a state-less context, the ViewState was introduced to allow for developers and users to have some semblance of history and state (Extreme Experts, 2007).

XML. XML stands for Extensible Markup Language. This language is made up of entities and optional attributes to represent arbitrary data (W3C, 2008). XML is designed to be easily read (or parsed) by computer programs, with many programs programmed to stop parsing the file if the XML is malformed.

Background History

In 2007, Kevin Matthews was trying to solve a particular problem of how to grade a number of assignments quickly while still giving students the proper attention they need when it comes to feedback. He developed the AGLS to ease his workload. The original version of the software was written as a desktop application written in Visual Basic. Kevin rewrote the application to work with the web and added requested enhancements so that more professors could use the tool.

Increasing the amount of feedback and its response rate was inspired by Robert Gagne's work. As part of his research into learning theories, Gagne heavily contributed to what is now considered the five types of learning and nine events of instruction

(Spech, 2008). The core concept with the research is that students learn through a variety of techniques.

A few different organizations are seeking to assist the students of today with learning various materials and improving their knowledge. The Science, Technology, Engineering, and Mathematics (STEM) Education Coalition works with a number of organizations, teachers, and students to ensure that the next generation is well adapted to succeed in the math and sciences (STEM Education Coalition). The Foundation for IT Education (FITE) has been working since 1975 to ensure that information technology is taught and spread to the many people and organizations (Foundation for IT Education, 2011). The Gateway Foundation wants "computers to be made generally available to underserved youth" to allow them to be introduced to a computer early on and start developing the needed technological skills (Gateway). There are countless other organizations – both profit and non-profit – that wish to help introduce skills and technology to the masses: this is just a small sample.

For the purposes of this project, computer skills are essential. It can be reasonably assured that computer skills are in demand for many jobs, even at a basic level. More students attend these classes, but there is not enough evidence that the number of faculty to cover these students has increased proportionally. Before the AGLS, more time was being spent on grading assignments rather than providing sufficient feedback or increasing course materials. As previously stated, multiple professors and assistants could potentially grade the same assignment but not with the same standards or rigor, causing inconsistency. There are a number of ways professors can normalize scores that are given when multiple graders are involved to minimize the inconsistencies, but it is perhaps better to avoid the issue entirely (Jacobson, 2001). While there are other issues also at

work, the potential for inconsistent grading standards is one of the primary reasons for the initial development of the AGLS.

The AGLS took the concept of grading digital assignments and enhanced it to make it easier for professors to use and to perform their job. Using the AGLS, professors have been able to grade a number of assignments and tasks for any Microsoft Excel or Microsoft Access document in a relatively quick manner. By allowing the grading to be done quickly, the possibility of professors taking on more students without sacrificing any quality of work or increasing the amount of assignments increases.

The concept of moving classes and assignments to an online format has been a recent trend. At UNCW, a number of the lower level Spanish classes are taught online with the help of the Rosetta Stone translation software (Vetter, 2012). Stanford University professor Jennifer Widom went from teaching approximately one hundred students in an introductory database course to almost one hundred thousand by moving her tests and lectures all online (Widom, 2012). Because there can be multiple students taking these tests online, it can be implied that there exists a way to grade all test submissions. Whether such a system can accommodate all of the students taking the class at once is beyond the scope of this capstone project.

Current System Capabilities

The AGLS is built from a number of components that allow the system to perform its needed jobs. While it is designed to allow additional components, there are four key components that must be covered. They are the Input Process, the Adaptive Grading Process, the Assignment Library, and Plagiarism Detection (Matthews, 2008).

Input Process

The AGLS is controlled through a web interface using any modern web browser such as Mozilla Firefox, Google Chrome, or Microsoft Internet Explorer. Through this interface, professors can create assignments, make answer keys, and grade the students' work. For future reference, the answer key or grade key is comprised of a list of tasks to be graded, along with their related correct and incorrect answers. It can also include any feedback provided by previous graders.

The key creation system should be flexible because there may be multiple ways to grade a cell or database field. Once the assignment and key are set up, it is possible to begin grading assignments. A common example is when a student, using Excel, presses a number key one too many times on the keyboard, missing the answer to a question (18 vs. 188). The professor, upon grading, sees this typo and is forced to count it as incorrect. On the screen where the answer is marked wrong, a comment box is provided for the professor to leave his/her comments explaining why the answer is wrong. A default comment is provided if the professor does not wish to take advantage of this feature.

Adaptive Grading Process

The ability of the system to adapt to a variety of correct or incorrect answers is one of the key benefits of the system. When students submit an assignment, their file (Excel or Access) is stored on the server. The professor can then grade all of the assignments with the click of a mouse button. At the start of an assignment's life, there is usually only a list of correct answers for each question. This is a direct result of the answer key that the professor generates. The AGLS parses the answers given by the students into two separate lists: one list containing correct answers and one containing incorrect answers. If an answer is identified in one of the two lists, the AGLS

automatically grades that task and moves on to the next task until all tasks are graded. If the answer is not found in either of the lists, the professor is prompted to decide if the answer is correct or not and to which list it should go. Once decided, the system remembers for the rest of the assignment's life and grades other identical answers in the same manner.

Assignment Library

Assignments created can be added to a library of possible assignments, and other professors can then use those assignments. A library assignment has tasks for grading the assignments with associated lists of correct and incorrect answers. This feature allows professors to have a number of assignments to assign to students, and to offer different assignments to different students each time the class is taught.

Plagiarism Detection

The AGLS comes with a built-in system designed to detect students that share answers or files. In Microsoft Access or Excel files, it is possible to easily embed hidden information inside the files. Microsoft Excel requires students to download a template to perform their work: inside that workbook is a hidden, password-protected sheet that contains a way of identifying the student. The AGLS can then be told to look for the hidden data to ensure that it matches the student who submitted the work. If there is a problem, the professor is notified of the situation. Microsoft Access relies on the timestamps of when tables and queries are created; this information is stored internally and cannot be altered by the student. For Access, there are a series of reports that highlight potential duplicate files.

It should be noted that it is not required for a professor to use or rely on the plagiarism detection system. If a professor allows students to submit their own work without starting from a template, there is currently no means to check for plagiarism.

General Procedure for Original AGLS Usage

The first step for professors to grade students is to let the Entropy grade book system know that they have a class. Once the students add themselves to the class, the professor is able to start grading the students' assignments. This can either be done from building an assignment from scratch, or by using an assignment from the library. An answer key consisting of known correct and incorrect answers for each task within the assignment can be made once the assignment is given.

Once the assignment is built and made available, the students can access the assignment. If required, the students can also download a template file from which to start their work. The students then complete the assignment and submit the completed files to be graded.

At any time during or after the assignment is due, the professor can grade what has been submitted. Assuming the professor has not changed any options for displaying the students' grades, students are able to instantly check their results once the professor has initiated the grading process.

Chapter 3: Methodology

There are a variety of system development methodologies that can be adopted when upgrading an existing system. Not all of the commonly accepted processes, however, may appropriate for the new features that are desired to be done for the AGLS. This section will discuss potential candidates for process to implement.

Waterfall

The Waterfall model of software development and most of its variants are designed so that all of the planning is done prior to coding. The programs that use these models are designed in one "direction" only: the Waterfall model is not designed for going backwards. As an example, if an important mistake is made during the requirements gathering stage and is not caught until the testing stage, it will necessitate going backwards to fix the issue before moving on.

Spiral

The Spiral Model allows for some of the same structure as the Waterfall model, but also caters to iterative development. Each cycle has four distinct tasks: planning the objectives for the current cycle, looking over alternative plans, developing and testing the chosen plan, and scouting ahead for the future cycles. The only major disadvantage with this model at this time is that the AGLS may not be appropriate for development. There is no budget for this project, and despite this newer AGLS being a graduate school project, it is relatively low risk. These factors can make the Spiral Model undesirable (Sparrow, 2011).

Agile

The Agile development process is another iterative development process. Agile development enforces the idea of communication between all members of the team and the customers outside of it to ensure a high quality product. There is built-in flexibility within agile development: this can allow for specific, controlled changes and features to be brought in. While it would be desirable to communicate with other programmers during the development process, the AGLS will, at this time of writing, be primarily written by one programmer. This lack of programming partners also disqualifies one of the more popular forms of agile development called Extreme Programming, for that methodology heavily emphasizes, and thus requires, paired programming.

Code and Fix

There is another "methodology" known called Code and Fix. It is regrettably a tried and true method of coding where a programmer adds code, recognizes that a bug is made, and then immediately works to fixing said bug with more code. Most of the time, there is no set plan or design when using Code and Fix. Because planning is required, it is best to avoid Code and Fix for as long as possible.

Summary

Considering that a number of features must be implemented with the new AGLS, an iterative approach such as the Spiral or Agile methodology will be beneficial. The specific methodology to utilize will depend on how much flexibility and adaptability is needed. If a consistent pattern is needed every few cycles, then the Spiral Model will be pursued. If adaptability to change is a more important focus, then the Agile method will be used.

Chapter 4: Outline of Completed Project

The rebuilding of the AGLS consists of multiple phases. First, there is a general overview on what the previous version was able to accomplish. Second, there is some discussion on some of the challenges in upgrading the old codebase. Third, there is a list of the features requested and the scope of the project. Finally, a few potential resources are identified to make building the new AGLS easier.

It should be made clear that just because the original AGLS supported a particular feature, does not mean that the new AGLS will support it as well. A general summary of what will appear in the newer AGLS compared to what got left out of the original package is outlined in Appendix C.

What the First AGLS Supported

It needs to be stressed that the AGLS is already a stable system. It has many features built-in that not only make the software powerful, but make it easy for professors to perform their grading work.

General Features

The current system is rather robust when it comes to features. Professors can manage multiple sections of a class at the same time, assign different tasks/assignment to different classes, and grade each assignment. The AGLS learns over time what answers are correct or incorrect. If mistakes are made during this process, the answer key can be changed, and re-grading can then be done for more accurate results. Partial credit can be given for answers that are “on the right track” but are not exactly the “optimal” answer.

The system also allows professors to provide feedback for the students about an answer. While the system automatically provides a general statement that an answer is

wrong, the professor can make custom notes explaining in more detail why an answer was not given full credit. This helps the students even more so as they can then correct their assignments and learn the material that much quicker.

Microsoft Excel Features

A number of features can be considered for Excel assignment grading. The AGLS is able to identify whether a cell contains one of the following: a formula, a number, or a text string. The contents can be checked for a variety of parameters, including cell formatting and value ranges. There is also additional support for grading charts and graphs.

Microsoft Access Features

Each submitted Access database has a number of gradable items. It is possible to grade the field types, the field names, the field sizes and the primary and foreign keys for any table. Inside each submitted database is a special table called “Queries”. Other gradable items for the table include the content of the fields to be displayed, the criteria (WHERE clause), or the input parameters.

Challenges for the AGLS Today

The AGLS was created approximately five years ago. This system has held up very well with mostly positive feedback. Still, it must be acknowledged that there are some areas where the system could be improved.

General Challenges

The core of the system is a website built in ASP.NET using the VB.NET programming language and Web Forms for the website. While the advantages of using Web Forms include rapid development and deployment, it comes at the cost of a lack of separation of concerns and great testability (see Table 1). The lack of easy testing does

not mean that testing of any kind is impossible: a concept known as integration testing is possible through automated browser programs (Selenium). While Web Forms can be unit tested to some extent, it would require moving code away from the Code Behind files and into separate files in proper locations. One would have to use a different methodology such as MVP, or Model View Presenter, to try to effectively unit test such a website (jminatel, 2010). When the system was first built, the ASP.NET framework was at version 3.0: it is now at 4.0. In addition, newer technologies such as MVP to increase code accuracy were not readily available.

Most of the professors that have used the AGLS found it easy to use. A few improvements were suggested, however. Dr. Judith GeBauer, a professor from UNCW, mentioned one issue that she had with the system. When assignments are created, she sometimes has a graduate assistant create the answer key filled with tasks. If a mistake is found in the answer key, she is unable to change the tasks. While there are some professors that would probably want to create a new assignment and answer key from scratch at this point, it would help if the AGLS made it easier to let professors know that they can delete individual tasks and simply recreate the one task rather than the whole assignment and key.

Another issue with the AGLS that some professors reported during the interview process dealt with the creation of the answer key. (Table 2 has quick summary data about the interviewees.) In order to create an answer key currently, the professor has to manually go through a form process involving many mouse clicks and text fields to fill in. These many clicks and text fields are used just for *one* answer (Kline, 2012). Considering that only one answer can be created per form per page load, the time it takes to create a key for an assignment is longer than it should be.

Another issue is that the AGLS is embedded into Entropy, a custom written grade book. While Entropy does help the professors, it is the embedded nature that makes the AGLS not modular. As one of the goals of this revision is to be able to market the AGLS to others, it will need to be re-written so that other institutions can use the software. The rebuilding will potentially allow for best practices that were not around at the original inception of the AGLS to be utilized to make a better product.

Microsoft Excel Challenges

Excel grading already supports a number of functionalities such as text, formulas, and basic graph reading. There are multitudes of graphing options available in Excel: not all of them are currently available for grading or feedback within the AGLS.

There are some issues that are not graph related. Primarily, there is no direct support for grading a scenario or solver. Instead, this data must either be manually reviewed, or a separate worksheet must contain the answers where the data can then be separately analyzed.

Microsoft Access Challenges

Grading Microsoft Access database files is a non-trivial manner. Unlike all of the other formats that were introduced initially with Microsoft Office 2007, the new .accdb format for Access databases is not in XML. Rather, it is in binary (Lagadec, 2006). This change of format makes it difficult to gather certain data from the database. This data includes forms, reports, and even the table metadata. MySQL allows for two commands to get table metadata: “DESCRIBE [tablename]” and “SHOW CREATE TABLE [tablename]”. PostgreSQL and SQL Server require performing a relatively complicated SQL query to get the requested data. At the time of writing, no solution has been found to grade Access metadata.

Technologies Chosen

In order to build the new AGLS, a number of technologies will be utilized. Listed below are the incorporated technologies, along with reasons why they were chosen.

Microsoft Windows 7

Both the Computer Science program and Information Systems program at UNCW have special agreements with Microsoft. One of those agreements relates to what software is available to the students free of charge. Thanks to that agreement, undergraduate or graduate students within the program are able to access a variety of tools and training through DreamSpark. While there are some restrictions to what can be utilized and for how many times, Microsoft wants DreamSpark to put "professional developer tools and training in your hands—all at no charge" to make it easier for students and professors to be more productive in the academic setting (Microsoft).

Microsoft Visual Studio 2010

As this project is a .NET web application and will be done on a Windows environment, an IDE, or Integrated Development Environment, that works in Windows is recommended. Microsoft Visual Studio 2010 is an IDE that allows for programming and creating various programs and websites in a self-contained manner. DreamSpark offers two versions of Visual Studio 2010, Premium and Ultimate. The differences between the two versions are irrelevant for development purposes: both versions offer a variety of tools that will be useful for writing this application.

The other candidates for reference were Visual Studio 2012 and MonoDevelop. MonoDevelop is a C# IDE that allows writing console and web applications on all operating systems, not just Windows (MonoDevelop, 2005). It was rejected primarily because of other needed technologies to be discussed shortly. As far as Visual Studio

2012 goes, that program was only released recently, and the 2010 version is available on all of the lab computers at UNCW. It will make things easier to stay consistent with the school computers whenever possible.

Team Foundation Server

One of the addressed challenges with the original AGLS was the lack of version control. Version control allows users and developers to keep a running history of the files changed in a repository to allow reviewing notes and, if need be, to go back to a previous version. While a number of version control systems are available, the decision was made to go with Team Foundation Server, or TFS.

A possible bad point of TFS is that it works best when using Visual Studio. Other version control systems do not have such a dependency, such as Subversion, Mercurial (also known as HG), or Git. However, this level of dependency is not enough to reject the choice to use TFS.

.NET Framework Version 4

The committee has requested that the new AGLS be developed using the latest stable version (version 4) of the .NET Framework at the time of request. While a newer version, 4.5, has come out since then, there will be no intentional upgrades to this newer version for this project.

One of the features that .NET 4 provides involves the use of optional and named parameters with functions. Optional parameters are parameters that can be placed at the end of a function parameter list after the required parameters, and given a default value. If these are not given values when the function is called via the programmer, the assigned value will be used. Named parameters will allow programmers to explicitly specify which values are being passed to which parameters in the order of the programmer's

choosing. This means that the order of the function parameters as originally defined does not have to be called in that specific order: the only requirement is that all required parameters are given a value. This can potentially allow for more concise code as the parameters are named when calling the function.

Another feature of .NET 4 worth looking into is the concept of Code Contracts. Code Contracts allow for enforcing a variety of code conditions such as pre-conditions, post-conditions, and object invariants directly within the code base (Microsoft DevLabs). This will also help enforce code integrity throughout the development process, and can help alleviate the testing process.

Visual Studio Unit Testing Framework

Visual Studio 2010 and the .NET Framework Version 4 provide a built-in unit testing system. The built-in testing system will allow for creating unit tests relatively quickly without the need to find a third party testing framework. To keep things organized, the tests will need to be a separate project inside the web application solution. There is a good chance that a models project will also be made for easier abstraction purposes.

The Visual Studio testing framework was not the only choice for unit testing. There are at least two other unit testing frameworks available called xUnit and NUnit. Both of these frameworks were rejected due to having the first party Visual Studio framework available. There is also a different style of testing available known as Aspect Oriented Programming, or AOP. AOP uses the idea of writing code such that cross-cutting concerns – logging and security are two examples – are handled by aspects instead of having this code located across a multitude of objects (Kiczales, et al., 1997).

While there is a dedicated aspect oriented framework available known as PostSharp, both personal and licensing reasons will prevent the inclusion of this library.

ASP.NET MVC 3 Web Framework

For this new version of AGLS, it was decided by the committee to look into a more current way of developing a web application that combines both the computer science side and the business side. The MVC web framework represents both sides neatly: MVC is a methodology used in many computer science fields, and frameworks are useful on the business side to reduce the amount of code that needs to be written. The other choice was to use the Web Forms framework: Table 1 compares the two in detail.

There is a possibility during the development of this project that a companion website will be made in a separate project that uses the more classic Web Forms style of development. If this is the case, the website will have to be made to use models more than the Web Forms' Code Behind system.

Razor View Engine

The ASP.NET MVC framework allows for a number of different view engines to allow rendering the final HTML to the browser. Microsoft chose to introduce a new view engine with MVC 3 called Razor that will be used for this project. Razor is designed to be more concise than the Classic View Engine's .aspx/.ascx files. The Razor files can also be more easily unit tested should the need arise to test the view files separately from the rest of the system (Gu, 2010).

Entity Framework

Entity Framework is a database framework that comes built-in to any new MVC based web projects in Visual Studio. MVC it is "an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects." (Microsoft)

What this means is that an individual can write simple code objects that act like database tables, and then perform methods and functions on these table objects to create, read, update, and delete the needed data. It interacts with the underlying database on the developer's behalf, generating safe SQL queries to send to the database. This reduces the need for raw SQL queries to be used with the framework.

LINQ

As defined earlier, LINQ allows for using a query-like syntax to retrieve information from a variety of collection objects. This can be anything from a list or array to even XML data or databases. The advantage of LINQ working with multiple collections is that it is possible to abstract a collection of model objects and use the same LINQ code on both the live data and test data, making it useful for unit testing. This means, among other things, that the same database is not needed for both development and production, and that two separate databases are not needed.

Entity Framework makes heavy use of LINQ. Specifically, it uses a feature known as LINQ to SQL to allow turning the LINQ code into the appropriate SQL code for the database of choice. When the data is returned, LINQ and Entity Framework convert the resulting data back into objects to be interacted with (Microsoft, 2010).

For comparison's sake, another technology that allows accessing a database to retrieve data is stored procedures. Stored procedures are designed to call the database once with requested inputs, perform a number of programming statements, and return a value once the work is done (Microsoft). It can effectively be used as another form of programming, only with direct access to the database.

Comparing LINQ versus stored procedures is interesting, especially when going into LINQ to SQL. LINQ allows for type safety and abstraction built-in. When

debugging an application, it is also possible to understand the generated queries easier. However, it must be acknowledged that LINQ is technically less flexible: stored procedures have complete access to a database's feature set while LINQ is not guaranteed that (Gillum, 2008). Similarly, there is no solid answer as to which technology is better in terms of security. Some say that stored procedures are easier to keep more secure, and others say that using LINQ to SQL already provides plenty of security since the parameters are treated for SQL injections by default (.net - Linq over Stored Procedures, 2009).

While one of the future goals of the AGLS is to allow for multiple database back-ends, this does not mean that stored procedures have to go away entirely. It is possible to not only use both LINQ and stored procedures at once, but also utilize LINQ to call stored procedures (Gu, LINQ to SQL (Part 6 - Retrieving Data Using Stored Procedures), 2007). To make this work best, any new database system that gets support in the future would either have to have support for stored procedures, or have the code be written in such a way so that alternate methods of getting and setting the data in the database are provided.

Microsoft SQL Server

Because the old version of the AGLS used SQL Server, the committee has requested that SQL Server is used for this revision of the AGLS as well. While SQL Server 2008 is available on the university campus and SQL Server 2012 is installed on the development machine, no 2012 exclusive features are planned to be utilized so as to maintain compatibility.

Other database systems exist such as MySQL, PostgreSQL, and SQLite. However, they were rejected for this iteration due to the simplicity of having SQL Server

access. Since the LINQ queries can be written in a way that all databases can utilize the data, future support for the other database systems can potentially be used with little configuration required.

Feature Requests

In order to determine where the focus should be for the rebuilding of the AGLS, a variety of ideas from many personnel were needed. Most suggestions came from a variety of scheduled interviews with professors. Some of the requests came from the members of Kevin Matthews' thesis committee that were within easy reach. Finally, some suggestions were either personal goals or ideas generated during some brainstorming sessions.

Version Control

While a number of professors requested version control, it wasn't made clear to all of them that a system was already in place for it. By using TFS for version control, this request was one of the simplest to fulfill.

Cleaner Code Base

A clean, consistent code base can go a long way to ensure higher quality in the long run. The old AGLS's code base is in its present shape due to the combination of the use of Web Forms and lack of testing. There are a number of items that are either duplicated or poorly documented that could be refactored. There are a number of database calls that can be optimized, or perhaps even removed once the simplified schema is in place. By also using version control above, it will be possible to track which sections get cleaner and more optimized.

Improvements to Microsoft Access

A number of professors have requested more abilities with the Access format. Access's format is hampered by some key issues. The first issue is that, unlike other recent Office 2007 file formats, the .accdb file format is binary and not XML (Lagadec, 2006). It is possible to get the resulting XML data of a table, query, form, or report by using Access 2007's ribbon interface. From the ribbon, an individual can get the table data, the schema of the table, and a presentation file to convert the data to HTML (databasedev.co.uk).

While this is beneficial for the old AGLS, there is a problem with the new AGLS. At this time, it is unknown if there is a library that can perform this operation without opening Access. Since having a library to parse the file without opening Access is required for quick grading, implementing this feature may not be possible at this time.

Additional File Parsers

At this time, the AGLS can parse Access (accdb) and Excel (xlsx) documents with a fair amount of success. A number of professors have requested that more file formats be supported. Not all of the requested file formats are feasible. Perhaps the only one that was possible to consider was one for HTML and CSS code. A few professors have requested the ability to parse HTML and CSS code to make grading introductory websites easier for them (Ferner & Patterson, 2012). This is perhaps one of the simpler parsers to consider as there are a number of libraries such as the HTML Agility Pack and TidyNet that can parse HTML regardless of how proper or malformed the resulting code is (DarthObiwan) (crocgod). For comparison, there are also web validators for both HTML and CSS available that can check the syntax for users automatically (W3C, 1994)

A number of requests have been received for file parsing that are not realistically feasible for the project. Ms. Marni Ferner of UNCW wants to be able to parse MySQL queries for her database class. However, as the AGLS is a Microsoft based system using SQL Server at this time, the goal of MySQL capabilities will make things difficult to test and verify (see the Technology section above). Dr. Laurie Patterson, an Associate Professor in Computer Science, wanted to look into a Java parser, but considering that current Windows operating systems generally do not ship with that runtime environment, there is no guarantee that it is available. Dr. Doug Kline originally had an idea of parsing SQL Server queries, but he has since changed his mind. It was later discovered that he wanted the ability to get a query's execution plan instead of being able to just parse the query itself (Kline, Ferner, & Patterson, 2012).

Non-Microsoft Office File Types

The committee wanted to know if other office suite file formats could be supported in the new AGLS to offer more flexibility. One of the file types that was considered was the LibreOffice/OpenOffice files. Officially, they are OpenDocument files (named after the original name for the suite, OpenOffice). The other file type of interest was the .numbers format for Apple's iWork's Numbers program. While some additional technical information is provided in Appendix A, a rough summary and background of the file formats are provided here.

Between the two "office" file formats, the OpenDocument format has greater support for future implementation. First, the LibreOffice suite is completely free, regardless of who is using it and for what purpose. Second, it works on any major operating system, from Windows to Mac OS X to Linux. Third, compatibility with the Microsoft Office formats is relatively high for Word and Excel. While the PowerPoint

compatibility is not as high based on information research, it is currently irrelevant for PowerPoint support as that is not a target for this project. Finally, a number of foreign governments have actually mandated the use of the OpenDocument format. Some of these countries include Norway (Orion, 2007), Hungary (Hillenius, 2012), the Kingdom of Belgium (Deckmyn, 2006), and many more. (Babcock, 2009)

The OpenDocument format uses a zipped archive; most of the files inside the archive are XML documents. This holds true for the Word, Excel, and PowerPoint equivalent programs within the LibreOffice Suite; the Access equivalent was not verified. The most noticeable difference between OpenDocument formats and Microsoft Office formats of comparable file types is that there are fewer XML files to parse for the OpenDocument formats.

Apple's iWork formats are also a candidate for future support, but actually supporting the files may be difficult. There are three programs that make up the iWorks suite: Pages is used for word processing, Keynote is used for presentations, and Numbers is used for spreadsheets. While the suite is not free, each individual item can be purchased separately from Apple's Mac App Store for a reasonable price. One disadvantage of the iWorks formats is that the programs are only designed to work with the Mac OS X operating system. The iWork suite also does not have a Microsoft Access equivalent.

During an interview with Dr. Thomas Janicki, it was first thought that the file formats for the iWorks files were in binary, for there was no easy way to open the files up at the time of the interview. Since then, it was learned that the iWork files are not actually files, but directories. Viewing these files on Windows Explorer, for example, shows that .key files are recognized as directories, and can be viewed as such. The .numbers and

.pages files, however, are not recognized, and required a separate application to open them. In this case, the 7-zip File Manager program proved to be useful in viewing the contents.

All three formats use a primary XML file for their content. Keynote first compresses its XML inside a .gz file: once again, 7-zip File Manager allowed for viewing its contents. While the file extension of the resulting file was not .xml, opening it in a text editor proved that it was XML. Pages and Numbers do not use a .gz file to compress the XML first; however, opening those files right away showed that all of the XML is stripped of unnecessary whitespace, thus putting it all in one line. This may be easy for a machine to read, but makes it difficult for a person to parse the files without the assistance of a style-adding, or beautifier, program.

Automatic Key Generation on File Submission

As it was stated earlier, the only way to create an answer key is to use the web interface, requiring many clicks of the mouse and keyboard for each task. This is particularly true for Excel files. Many of the interviewed professors have requested an easier way of generating a key, via submitting an Excel file. Upon submitting an Excel file, the AGLS would then have to be able to extract the relevant data from the file to form the initial key.

Web Service for Quicker Grading

As a number of students have completed the same assignments, many right and wrong answers have been recorded by the AGLS. For some assignments, it is completely possible that a professor can click on the grade button and not have to make any changes whatsoever. For situations such as these, it should be possible for students to submit files

and have them automatically graded. This would require implementing a new web service just for the specific assignments.

As this web service would not have professor be required to click a button to grade assignments and go over unknown answers, there needs to be a way to address what happens if a wrong answer is found. One possible option would be to allow for a third list, one consisting of unknown answers that are uncovered. Any answers in here could be emailed to the professor to let them know that an unexpected answer has come into the system, with an optional request to look into grading the unexpected answer as also correct. The student could also be emailed about the discrepancy.

International Language Support/Localization

As this new AGLS is designed to be marketable to a number of colleges and universities, it needs to be flexible. There is always the possibility that future customers for the system may come from a country that does not speak English. As such, it would be helpful to keep open the possibility of supporting multiple languages within the new system. Some approaches that can be taken include, but are not limited to: sessions (Adamyan, 2010), custom route handlers (Adamyan, Addition to ASP.NET MVC Localization - Using routing, 2010), and cookies (jgauffin, 2011).

Support for Multiple Database Back-ends

At this time, the AGLS uses Microsoft SQL Server as its back-end. As LINQ will be used to write the database queries, support for other databases is possible already. However, it will not be pursued further than that for this project.

Scope

There are three levels of scope for this AGLS project. The first level deals with items that, barring extenuating circumstances, have to be implemented. The second level

contains feature requests that have been requested, but not required/scheduled to be implemented. The last level deals with any future requests that, barring extraordinary circumstances, are not going to be considered for this project.

Project Scope

This project is intended to be the start of the new AGLS. As such, the goals listed below are only for the upcoming semester. More will need to be done in the future to make it as feature complete as the current version while also offering more flexibility for the programmers. These items are listed in order from highest priority to lowest priority.

Rebuilding from Scratch

One of the primary goals of this project is to remove the present ties between AGLS and Entropy. This will require starting from scratch and building a new optimized database schema. This system should primarily be a grader with the ability to export grades in a number of compatible formats. This will primarily be done using the MVC3 web framework.

Adding Unit Tests

In order for this to be commercially viable, the software needs to be tested. The previous version of the AGLS did not allow for easy unit testing. While integration testing was possible, no tests were noticed that utilized them. Because this software is being rebuilt from scratch, it will be the perfect time to properly test as the code is developed. The unit testing will be focused on specific core models of the project. Unit testing the web pages (the views) should not be required as integration testing can be done. However, the technology that will be used will allow for unit testing the pages if it becomes required.

Adding Excel Key Submission File Support

Almost every person that was interviewed stated that setting up answers and tasks for the grader was the most difficult and time consuming part of creating assignments. It takes many text fields and form options, not to mention mouse clicks, to enter the correct key for one single answer. Many of the interviewees have requested that there should be a way to simply submit an answer key in an Excel file.

As answer key submission was one of the most heavily requested features, it would be a disservice to those interviewed to ignore the request. As such, there is an obligation to make this request happen. In order to do this, the AGLS will have to parse an Excel file based on what is submitted and generate a basic key from that. Each cell's contents and formatting would all be designated as potentially gradable answers. When the file is parsed, a web page could show up explaining what the AGLS determined as items that could be graded. The professor can then add, delete, or change any grading tasks, as well as assign a different number of points for each task, at this point.

Allow Web Service for Interactive-less Grading

Some assignments will be composed of questions that only have one answer. Other assignments may have had many answers come in over the semesters and stored in the lists containing right and wrong answers. For assignments such as these, it should be possible to create an option to allow for automatic grading of an assignment upon submission of an assignment.

Such a feature should be made an option for each individual assignment. If the professor decides to use the web service for grading, the system will not be able to learn: if an answer is not located in either list, it will be counted as incorrect. The professor would have an option upon enabling the web service mode if he or she would like to be

emailed upon receiving any answers that are not known right away as correct or incorrect. Should any submitted assignments require a re-grade as a result of the newly determined correct answer; the professor will still have that option.

Add General Support for Exporting Grades

This new AGLS is meant to be a grader, not a grade book. While there will be some information about the class role and what assignments each person has, overall grades are not a part of the scope here. As such, the ability to export some or all grades will be essential.

The new AGLS should always be able to export grades to the XML and JSON formats. The CSV format should be possible, provided that the comma symbol is not used as the delimiter. The reason for that is due to the nature of text answers for Excel programs: there is a chance that some professors will want to see commas in an answer, and as such it is better to use a non-printing character such as a tab to delimit answers to avoid a conflict down the line.

Possibly In Scope

The following items are ones that would be great to have, but have been classified as low priority for this project. Any item listed in here is not required to be done for completion of the capstone. Any items that get done in this section will likely be from the highest priority (the top), and then going down the list as appropriate. Perhaps future research will see AGLS expanded.

Adding OpenDocument Spreadsheet File Parser

As previously stated, the OpenDocument suite is used by a number of foreign governments due to the required format they must use. Furthermore, LibreOffice is free for everyone regardless of purpose, and is available for all operating systems. It has also

been stated that Microsoft Excel has been the primary focus for this project. If any new file parser is to be added, it should be the OpenDocument Spreadsheet format. Some of the format's details are in Appendix A.

Add Specific Software Support for Exporting Grades

In addition to the CVS, XML, and JSON formats, there are also specific formats that some established grade book software already has in place. One of the products that a number of UNCW classes use for a general purpose grade book is called Blackboard Learn. As many instructors prefer to use Blackboard Learn for grading, it would be beneficial to add in support for exporting the automatically generated grades to Blackboard Learn or other similar software. This will help ensure that the new AGLS becomes a competitive product.

Out of Scope

A number of items have been considered during this process, but not everything can be done within a single semester. This section catalogues what is known to be out of scope for this project. These are classified according to subject material.

Other Microsoft Office Formats

Not all of the Microsoft Office formats will be immediately supported; only Excel is definite. Other formats will need to be targeted for later versions should demand arise.

Microsoft Access Support

The .accdb format is used in the older version of AGLS, but comprehending its format is still very difficult. It does not help matters that this format, unlike the others introduced around 2007, is a binary format. As such, there will be no support for Access in the new AGLS.

Microsoft Word and PowerPoint Support

It is theoretically possible to parse the 2007/2010 Word and PowerPoint files through the same means as Excel. The means technically exist: there are already programs out there that can analyze what is written and grade a paper relatively quickly, though some opinions differ on whether or not that is a good idea (Thomson Reuters, 2012). The obstacles that have prevented looking at these formats at this time include the original AGLS not grading these formats and lack of time to research their particular formats. As such, parsing these two file formats will have to be delayed until a future version if there is a demand.

File Formats not Based on Office

There are more file formats out there than just the Office based formats. However, it would be non-trivial to support these formats. The Numbers format from Apple's iWork program is not being supported: View Appendix A for more information. Java file support is not going to be included since Java is not guaranteed to be present on all Windows machines. Database files (regardless of database system such as SQL Server, MySQL, PostgreSQL, etc.) are not being included due to both disagreements on how the files should be graded and the not-guaranteed availability of the database systems on the Windows machines. Finally, HTML and CSS files can be parsed, but would work best if the professor could view the pages in a web browser to view the grade.

Non-File Format Features

Not all of the feature requests that were made are related to adding support for new file formats to parse. This section deals with the requests that do not involve parsing a new format, but still had to be turned down for one reason or another.

International Language Support/Localization

As this new AGLS is intended to be marketable, it may eventually need to support people that speak a different language than English. While learning another language to have a better understanding of a country and its culture is admirable, it is not feasible to do so in a timely manner.

Support for Importing AGLS v1 Libraries/Assignments

As this is a new version of the AGLS, there is no guarantee that older libraries and assignments will be compatible. During the development of the new AGLS, the old one will still be used, serving professors and students. During meetings with Kevin Matthews, it was determined that adding such a feature would not be worth adding at this point primarily due to the plan of adding Excel key submission support. Perhaps in the future, there can be a way to import assignments that were exported from the original AGLS.

Support for Multiple Database Back-ends

The committee for this project has requested that the first re-visioning of the AGLS is to stay on SQL Server for the purposes of this project. As such, the feature request of adding support for other database back-ends such as MySQL and PostgreSQL is automatically disqualified. However, the code that needs to be written can still be done so in such a way that it will be easier to support multiple databases in the future. One of the best ways to look into this will be to embrace the LINQ technology.

Planned Resources

In order to use the technology effectively, reading about them will be required. There are a number of resources and reference materials regarding the technology planned for the AGLS. Fortunately, reading each piece in their entirety will not be required for the most part. The lists below are not final: more materials may be required.

This should give, however, a solid idea of what materials are available now, and what could be discovered in the future.

Skillsoft Books 24x7

There is a service, known as Skillsoft Books 24x7, that allows individuals and companies to have access to a huge library of books and summaries on demand (Skillsoft, 2011). UNCW has access to this service through their own Skillport service. A list of the books planned can be found on Table 3.

Selected Web Resources

While there will be more web sites and resources found during the course of this project, there are a few of particular interest that are worth studying. This merely mentions two of the pages.

One such page is Microsoft's own documentation library (http://msdn.microsoft.com/en-us/library/gg416514%28VS.98%29.aspx). Inside the documentation, it talks about the different components of a traditional MVC application and how to apply features such as filtering and unit testing to one's app. The library also makes mention that ASP.NET MVC is not going to "replace the Web Forms model": both are to be maintained in the near-foreseeable future (Microsoft, 2012). For reference, it is possible to not only have separate web projects for both MVC and Web Forms at once, but also have a core MVC application that contains Web Forms pages, and have both frameworks take advantage of common functionality (Kaimal, 2010).

Microsoft also has its own resource website (http://www.asp.net/mvc), where an individual can access sample code, tutorials, and videos (Microsoft).There are more features and articles available to learn about MVC than what can be talked about in this project.

Chapter 5: Building the AGLS

Planning pales in comparison to creating the actual product, however. Some last minute items needed to be addressed before the final building of the AGLS could start.

Practice Solutions

Building .NET applications is not new to the author: a direct independent study was taken on the C# language. During that class, a number of example programs were made, including one that involved VB for expanded experience. For this project, new example programs were created, but this time with a focus on the web related programs and technologies.

One of the first goals was to identify if there was an existing library that could handle the parsing of 2007/2010 Excel files. That goal was accomplished quickly: a library called EPPlus existed. EPPlus, most likely short for Excel Package Plus, exists to allow users to programmatically read and write Excel files in the new format (Källman, 2009). To make things better, the library is also open source and under the LGPL (meaning it can be used commercially).

As a side task, some code was written in a public fork of the EPPlus library to try to add a new feature to the library. At the time of this writing, the fork is under a pull request for Jan Källman and his staff to review the material to see if it can be included. Even if this request is unsuccessful, there are other ways, without needing to modify the library, to get access to the requested data.

The second, and largest, program that was built was the famous example known as Contoso University. Contoso is designed to teach programmers how to use the Entity Framework and a Code First approach to build database tables out of code objects and then letting the framework handle the table creation (Dykstra, 2011). There are ten pages

that cover more than just the Entity Framework, however. The lessons include sections on table inheritance, repository patterns, and including a library via NuGet, the package manager system.

Other example programs were created to ensure that programming skills were up to par before building the primary application. First up was a simple program designed to connect to a personal instance of SQL Server. The main thing learned from this experience is a reminder that, similar to the .NET Framework, Entity Framework uses convention over configuration many settings, including the names of database context connections. The name of the connection must be a specific name depending on the name of the context class; otherwise, a default database is chosen (raduenuca, 2011). After that program came two seemingly simpler programs: one implemented SASS using the Mindscape extension, and another showed how the Entity Framework can generate models from a database if required.

Three more test programs were written before the core project's code was first made. The first consisted of a lesson in dependency injection. The code was written successfully, and the tutorial that was followed helped explain what needed to be done. The ASP.NET MVC framework had work done to better support IoC, and Ninject, the DI library used, took full advantage of it to provide a clean way for injecting objects on runtime (Khan, 2010). The next program was supposed to teach about test driven development: unfortunately, that tutorial did not explain things well enough during setup or execution. The basic concept was grasped, however. Lastly, work was done with the ASP.NET Session object to ensure that a backup plan would be available.

Building the Application

The work on the new AGLS started with a blank slate: an empty project was made, which left the bare essentials. In short order, the built-in packages were updated to their latest versions via the NuGet Package Manager Console and some simple pages were set up. The layout of the page was copied from the Contoso University example, though the CSS was turned into SCSS and the background color was made teal to match UNCW's colors.

While in development, the first of the model classes meant to represent tables was created. A connection string was set up with starting data was seeded. That does not mean that the database was automatically loaded upon clicking a link to the page. As it was discovered later, the tables are made and initial data inserted whenever the first call to the database is made. If a page does not require a database call, the database is not touched.

Next, an email handler was needed for the purpose of a contact form. It was missed during the initial planning that registering to a website for a service usually involves an email. Thankfully, such an email service called Postal was found. The service was not only relatively simple to understand, but also had a compatible license for development and production. Two emails were crafted with unit tests with passing results. When it came time to actually use the form, the emails were "sent" successfully. The reason for the quotes is due to the development configuration. Currently, any email that is generated gets placed in a specific folder of the hard drive.

The only problem that was discovered with Postal is something that Postal, or any email library for that matter, cannot control. When emails are saved to the hard drive, the programmer is not presented with a way to name the email in any particular way. Instead,

a filename is chosen for the programmer using a GUID, which is a globally unique identifier. It could not be confirmed in a timely manner if this is related to an email standard that requires emails in file systems to be named in a certain way.

Another important item was learned during the using of Postal. Whenever a form is submitted in ASP.NET, the results of the form can come through one of two primary means. The first way is to use a general FormCollection object that contains all of the form fields submitted (Microsoft). FormCollections are available to all forms submitted in any ASP.NET system, not just MVC. The second way is to create an object that specifically has the same property names as the form elements names: this is considered to be a strongly typed object to the form. The advantage of the second way is that it is possible to add attributes to the properties to specify what items are required in the form, among other validations. This technique is used often in Contoso University with the model table classes, but it is not required for the class to represent a table in the database for this to work.

The next big hurdle involved testing authentication. Traditionally, this involves using either the default MembershipProvider that comes with a non-empty ASP.NET MVC setup, or creating a custom MembershipProvider. For this project, a custom provider was created that extended from the base MembershipProvider to only support the features needed and not be reliant on a provided-by-default database (Harman, 2011). What ultimately helped was being able to use Ninject to inject properties to the providers to allow the providers to work when called upon automatically (Gloor, 2012).

Even with implementing the provider above, there were bugs across the models that needed fixing. First of all, not all tables use a single primary key: some prefer a compound key of two or more columns. If this is done, then it is important to specify

what key would normally come first for ordering purposes (Mrnka, 2011). Second, two of the models were set up in such a way that they had virtual instances of the other. This would cause a problem if anything needed to be deleted: the operation would never terminate. Code was added to prevent this cascading cycle (Kamyar, 2011). Lastly, an error in the setup of the code revealed that the encryption method in place was returning twice the number of characters than originally anticipated. Because the code was crashing during the seeding (model validation takes place even here), exceptions and breakpoints were used to confirm the extra characters (Slauma, 2011).

Once the bugs involving Entity Framework were eliminated, the membership provider performed its job correctly and allowed authentication of users. The backup plan of using the ASP.NET Session for authentication is now no longer needed. There are more ways to utilize the Session than just authentication, however.

During some light browsing, a different tutorial explaining different parts of implementing a website with MVC was discovered. Called the Music Store, it crossed over into some of the same topics that Contoso did, but also went more in-depth with user registration, shopping carts, and jQuery (Galloway, 2011). The idea of being able to implement a shopping cart now instead of waiting would provide a head start on a long reaching goal of this project: marketability. In order to even complete the cart, however, registration and logging-in needed to be addressed.

Earlier in this process, it was not known if users to the website would have to register first before being able to buy a class or another service. The Music Store showed a way to allow potential customers to browse and add items to their cart, and then have the cart migrate to the user upon logging in or registering. This is where the ASP.NET Session is coming into play: it stores a GUID that contains that session's cart. Unlike the

tutorial, however, the AGLS implementation of the cart keeps the GUID even upon transfer to an authenticated user. The Music Store does not explain well, however, as to why to change the cart identifier from a GUID to a username when GUIDs can be useful as database keys, regardless of whether they are generated randomly or in a sequence (Perkins, 2010). By keeping the GUIDs in place, this allows for functionality such as multiple cart history for the logged in users and being able to restore a cart upon logging in. Unfortunately, the shopping cart state can possibly go out of sync. It is not clear what will happen to the state if items are added to a cart while not logged in and then the user logs in, only to have a cart of different items and quantities.

One of the final actions involving the tutorial cart required the user to check a box saying the items were free. While no money should be spent right now due to the AGLS being in development, this does not mean that the program should not prepare for the eventual case. A checkout page was made where, instead of asking for shipping information (as a reminder, anything to be sold in this AGLS is digital), a form is available that simply asks for credit card information. Validation is handled through a custom attribute. Any validation for a field can take place by extending the ValidationAttribute class and overriding the method named IsValid (Allen, 2010). In this case, a credit card validation attribute was made to ensure that the string of numbers was between 12 and 19 digits in length inclusive. While it may have been better to simply use a regular expression, the concept of extending an attribute would come into play again.

During the last testing of the cart, the final operations were to remove an item from the cart and add an item to the cart. Removing an item was handled via an Ajax request in the Music Store: adding was not. Due to how the AGLS website would work, both were made with AJAX functionality in mind. To leave the possibility open for users

without JavaScript down the road, an Ajax Attribute was made to allow easier splitting of logic down the road. In this case, all it took was extending the ActionMethodSelectorAttribute and overriding IsValidForRequest (Leeks, 2011). Should there be other Ajax requests needed in the future, the attribute is now ready.

Unfortunately, working on this cart revealed a bigger issue with regard to the Entity Framework. During the calls to add/remove items to/from the cart, two different repositories representing the tables were called. While the repository pattern is a good one to use for database access in general, it can get be difficult if two or more repositories with different contexts try to access the same object either directly or indirectly. To paraphrase what Rick Strahl has written, the LINQ to SQL mechanism not only uses a database context to access the database, but also keep track of the state of the objects affected. Not all ORMs do this, but LINQ to SQL is an exception. Thus, it assumes that there is only one context accessing the database (Strahl, 2008).

With the problem established, the solution seemed simple: find a way to have all of the database repositories share a single context. In other words, the goal was to aim for "a DataContext per user, per post back." (Meacham, 2009) What helped is that there is another collection of sorts, similar to the Session that allows for storing anything for a per-user basis. The HttpContext object contains a "bag" of Current Items that is tied to the individual user, but only lasts a single request. With this in mind, it is possible to utilize a pattern where an object watches over other objects to avoid concurrency issues. This is the Unit of Work pattern (Miller, 2009).

A Unit of Work "store" was created that can watch over any objects that come, for lack of a better term, visit the store. However, passing the database context to it was not enough. There needs to only be one instance of the context existing at all times. As

such, the database context needs to be under a singleton pattern. The key difference is that “we don’t use a private static field to hold our singleton instance, we use the UnitOfWorkStore class.” (Meacham, 2009) A singleton Layer class was created to maintain the single instance of the database context.

Even with the singleton made, more work needed to be done for the repositories in use. At the time, a GenericRepository pattern was in place to simplify the code writing. More specifically, a Repository was in use that required a new database context each time it was created. As that was unacceptable, a new generic repository was created for this project that took advantage of the singleton. Creating the code was not tricky: the original version was under open source, available for anyone to take and modify. The licensing term for the GenericRepository is the Microsoft Public License, which seems to allow modifying anything inside the code base as long as attribution is given.

At this point, it was time to start working on the file upload process. The first step was to identify how to properly handle file inputs. There is a class called HttpPostedFileBase that already allows access to an uploaded file’s content length, type, and more (Esposito, 2010). Thus, that code was put into a form model to be validated. To further ensure that only 2007 Excel based files are uploaded (well, the non macro ones for now), a custom attribute was written once again to ensure the specs matched (Dimitrov, 2011). One of the specs that needed confirmation was the content type. Each of the new file formats not only had a specific file extension, but also had a standard MIME type that web servers should recognize on the web (VsOfficeDeveloper, 2008).

Once the preparations were set up, the testing was short and relatively quick. The file was sent to the specified folder and the browser showed a status message. Since the files were technically in a potentially web-accessible directory, a route was created that

would theoretically send malicious users to an error page. That did not work as intended: instead, a 404 standard error type page showed up. The plan of setting up a dedicated 404 page has been abandoned indefinitely to focus on the necessary tasks to get the AGLS running as it was supposed to.

Part of the purpose of this application is to track when Excel assignments are submitted. This required another model. It was originally called File, but due to File already being claimed by Microsoft, the local model was renamed to UserFile. The use of attributes allows for the database table to be called File without conflicts with either the database or the C# language.

Tracking the files was not difficult. Getting the files back was more difficult. No matter what was done, the routing section of the Global.asax file needed a new route to handle a download. Since a DateTime object can be converted to binary (technically, a 64 bit representation of the DateTime, making it a “long” data type), the parameter to be passed in would be a numerical representation. Some adjusting of the DateTime and String extension files was needed to make sure that the formatting of the strings and dates was consistent for both upload and download. Downloading was performed with a FilePathResult object to allow writing the file without needing to use the server’s memory for the job. A FileStreamResult object is also possible, however, if reliability is essential (Brind, 2012).

Once the file download worked, a constant file was made with a static class to contain the MIME type. This is a form of abstraction and simplification. There is no reason for the bare string to exist in more than one place when a constant file can contain it, allowing for fewer errors.

With the first file upload test completed, the AGLS's cart system was re-investigated. There were two primary issues that needed resolving. First, the carts needed to be loaded from the User's Identity variable first rather than the session. This way, it would always be easy to get the latest proper cart for a User. Second, a caching issue needed to be addressed. If a user "purchases" any items, and then navigates their way back to the store page to purchase more items, the old information displays. Upon clicking on a link to add or remove an item, the values fix themselves, but that defeats the purpose. The server needs to be told how to cache – or rather, how not to cache – certain pages so that data is always loaded fresh from the server rather than loading a local copy from the user's computer (Dighe, 2005). Once the cache disabling code was put in, the pages worked as intended.

The next step was to prepare for adding a class to a user that bought classes. This required a number of select lists and Ajax calls. The select lists were handled by an extension method that allowed turning a LINQ query into the appropriate select list, albeit with modifications to the original code to allow for more flexible options (Stock & Krome, 2009). The reason for the Ajax calls was to avoid pulling all of the data at once when it was not necessarily needed: only the list of countries was guaranteed to be around for the entire page. However, a problem was still uncovered: unlike the prior Ajax call, the ones here originally did not work. There is a security feature built-in to the framework where Json post requests work normally, but "get" requests require an extra parameter when forming the JsonResult in order to bypass the restriction (Allen B. , 2009).

Adding a class was not yet done, however. While an ASP.NET MVC application by default comes with jQuery UI, and thus a date picker, there was no equivalent time

picker. A time picker had to be found, and it is fortunate that one was found and implemented to some degree of success. (see the *Making Tough Decisions* section below) Once the library was implemented, only one more custom attribute was needed: in this case, one to ensure that the dates lined up and did not get entered out of order. The results proved satisfactory.

During the development of one of the more complicated tasks, it was realized that the code written has inconsistent documentation in many places. As one of the goals of this AGLS is to provide good documentation and current best practices, the StyleCop extension was installed. StyleCop works by analyzing the source code that was written to ensure that a set of rules for writing clean consistent code are followed (andyr, 2012). A number of rules were already pre-set, but all could be turned off if needed. Only a few rules needed to be turned off for a variety of reasons: comment headers were never used much, the #region pragma is honestly useful for folding away code that does not need to be seen at all times, and the rule about not using tabs could easily be worth another paper in and of itself. With the StyleCop plugin in place, some of the classes have already received the consistency treatment. More diligence is required in order to not be lax in writing clean code and documentation.

With a formatter in place, it was time to tackle another issue: reading the Excel files and retrieving data. The first test was to write a simple view model that would contain a list of the names of the worksheets, and then print them out in a list. That part was relatively simple to implement. The next idea was to try to copy the entire ExcelWorkbook object into the view model, and then output its contents by going through the objects properties. That did not work as well due to the fact that the underlying ExcelPackage is IDisposable.

To be IDisposable means that the class is capable of clearing out, or freeing, its own resources (Krysmanski, 2012). The resource in the AGLS's context is an Excel file. One of two coding patterns can be used to read IDisposable resources: either the traditional try/catch block with a finally, or a using statement (Microsoft). No matter which pattern is used, the same thing happens: the pattern opens the resource, and closes the resource when done. It is this closing where the problem came in.

The ExcelPackage is Disposable: when the package is closed, all of its objects are released. This includes the ExcelWorkbook and whatever children it has. The assigning of the ExcelWorkbook into a variable was done by reference: this means that the view model was pointing to the same instance instead of having its own independent copy (Albahari). Thus, the view model was pointing to an unavailable resource when the file was closed. This was worked around so that the reading of the excel file took place on the view itself. This is perhaps not the best idea, but at least it worked for the time being.

A minor break was taken from working with Excel items to pursuing a concept known as remote validation. Remote validation is when an asynchronous call is made to the web server upon a changed value in the form to validate it. Unlike prior versions of the MVC framework, the only configuration that is required is the addition of the Remote attribute to the property to test (Stannard, 2010). Remote validation ends up being useful to validate items that require using a web server, such as duplicate value checks.

A minor caveat is that remove validation requires JavaScript to be enabled in order for it to work. This means that any server checks done on the Ajax call will most likely need to be replicated during the normal submission unless one wants to force JavaScript on users in order to do anything. For now, this is not an important enough item to worry about.

The next step is to set up the database tables relating to a classroom. Using UNCW's system as a guide, there are a number of courses inside a university or school. Each course can have any number of sections, and each section is active for a particular date range. There is no department layer within this system: one can be added later if there is demand, but it did not feel needed. During the creation of the form that would create such a class, a new feature was required: hiding a property form value programmatically. A hidden attribute was added to the appropriate input elements to hide values that still needed to be submitted (meisol, 2010).

Once those tables and forms were set up, it was possible to focus on a new controller, a Class controller. This controller is designed to manage all aspects of the class. A teacher entry page and very basic assignment form page were created right away. To utilize the form page better, due to the URL structure assigned at the time (it takes 4 parameters: a school ID, course abbreviation, a section name, and a formatted date of when the section starts), the ASP.NET Session was once again employed. In this case, it was important to ensure that the Session was set before accessing the form page. Utilizing an action filter attribute and its filter context allowed for the proper redirection in a clean manner (ŁukaszW.pl, 2010).

Both the assignment form and the subsequent answer check form needed advanced techniques in order to render them cleanly. The goal for both forms was to utilize the Html.EditorForModel() method to automatically generate all of the form fields based on the properties of the form object. At the start, both forms would not generate all of the data. The assignment form would not generate the input field for the HttpPostedFileBase object, and the answer check form generated nothing due to using lists of complex objects. There is a default model binder within the MVC framework that

can bind nine different data types, not all of which are variables (Wilson, 2009). Unfortunately, complex types and collections are not a part of the nine, and thus require extra work in order to render them properly (Haack, 2008) (amit_g, 2011).

In order to address both issues at once, it was imperative to understand how binding worked. To bind complex objects, special partial views (or templates) had to not only been written in code, but also placed in special folders. Inside either a shared views folder or a specific controller's views folder, one can write either a DisplayTemplate or EditorTemplate that can control how a type renders to either an HTML form or just a display page that talks about the core model (Jones, 2010). It is possible to even change how existing types display for either viewing or editing: there are six steps that indicate which template gets called (Wilson, 2009).

With those six steps in mind, it was easier to understand how the default model binder performed its job. The source code to the framework revealed that, by default, there is a maximum depth of 1 for all objects in the form: any that are beyond that depth are typically rendered in a simple way (Headcrash Industries, 2011). Armed with this new knowledge, a new default template was made for the two forms. The key differences between the built-in one and the one written are that the written one allows complex objects, and the objects can be at any depth. These changes allowed for writing a proper model binder for the HttpPostedFileBase object, thus solving the issues with the assignment form.

The answer check form proved trickier. The original planned used five template files: one for the original object, two for collections of objects, and two for the objects within those collections. When it came to viewing the form, the form elements were being generated, but with improper id and name fields in the HTML attributes. This is

attributed to how the framework handles any calls made to all of the Html.<Whatever>For calls. Inside the method is a lambda expression indicating what part of the model to render in the preferred way: it is the expression itself that determines how the attributes are rendered (Kondo, 2010). This does not mean that one is stuck with these defaults, however. There is a property in the ViewData's TemplateInfo called HtmlFieldPrefix that indicates what the prefix currently is for the present object. Comparable to almost everything else about the framework, it is possible to customize this value to work with whatever objects are in play (counsellorben, 2011). There are a number of ways to customize this: it can be done via splitting up the prefix into individual parts separated by periods, or using a regular expression to remove unwanted data (Burnett, 2011).

Fixing those issues only revealed more problems unfortunately. The first one was relatively simple: assigning a CSS class to a form element programmatically cannot be done with the word "class". Class is a reserved word in a number of languages, including C#. Luckily, prefixing the at-sign (@) character before the word fixed that issue (Duffy, 2009).

The next issue dealt with the use of the decimal type. An attribute was added to the complex form object to ensure that a decimal value was a non negative number. However, said value could not be passed in the attribute's constructor due to how decimal numbers are handled. Attribute constructors can only take simple values and strings: decimal variables require instantiating a new object, and objects cannot be passed in since the constructor call is a method call (Hare & ldsenow, 2009). In this case, the workaround to utilize was to pass a string value that resembles a decimal value, and then parse it at the end.

The third issue was only encountered due to the contents of the Excel file in use for testing. Said file contained three of four spreadsheets, two of which were the result of automatic generation of a solver report. This meant that a huge number of form elements were being sent to the server for model binding. After an arbitrary number of form elements, it crashes with an InvalidOperationException (Hollhumer, 2012). Having to deal with all of those form elements caused a separate issue: the Firefox browser was also performing client side validation before sending the data to the server, and current browser settings have the browser hang for roughly ten seconds before being prompted whether or not to continue executing the client side validation (Firefox, 2012). Since there can be no assumptions on how users have configured their script settings – or any settings for that matter – this can potentially mean that the client side validation may have to be disabled in the future.

To avoid frustration, a new part of the website was worked on soon after: adding students to the class. A professor needs to be able to use the system to add anyone to their class as a student in order for this system to work. Unlike the answer check form, this form would not need to extend any of the custom templates, for everything is being rendered by the same file.

Up to thirty students at a time can be entered in the form. For students that already attend the same school, a drop down menu is available to quickly select that student to add to the class. This required using the except LINQ clause to perform a set difference operation (Horst, 2008). Should the student in question not be on that list, most likely due to not having used the AGLS before, the professor can supply the known information about the student. Regardless of whether the student is new or existing, an email is sent to them using the Postal service.

Getting to the point where Postal could work had its own issues. This form was built manually for the most part: while the helper functions were used, a custom function was needed to ensure all of the form elements' name and id fields matched up with how C# expects them. When mistakes were made with these fields and the model state did not validate, it often took investigating and iterating through the individual properties of the model state to indicate what was going wrong (Elliott, 2010). The individual errors were also investigated by going through the keys of the dictionary-like structure (McKenzie, 2010). After a few iterations, the form was completed enough to focus on the next task relating to the form.

As this form is designed to add students, the students need a way to view their class. This required making a specialized student index page for the class controller, and duplicating a number of models and functions to get class and assignment information that is tailor made for students rather than teachers. More initialization data was added to allow for quick testing. Thankfully, the data was put in correctly and the pages rendered properly with what was supplied.

Focusing back on the Excel parsing, a few items were addressed to make the transition easier. First, it was decided to break up the grading process: there would be one form to handle summary information about the worksheets, and a second form to handle the answers regarding an individual sheet. The functions were renamed to allow for both forms: unlike normal object-oriented programming, the MVC framework uses attributes more than method parameter lists to determine which method is called (Levi, 2009). Answers for the individual sheet were made read-only, with only formula and value answer types possible at the moment. Enumerations were created for drop down purposes, and hooked into the forms via an extension method (Goldstone, 2011).

While the forms were being created, an interesting issue arose. None of the tables in use are assigned to the default schema of the database. Every database system has a different way of handling default schemas: with SQL Server, the default schema is usually called “dbo”. As explicit schemas are in use due to attributes, this means that none of the tables are using the dbo schema. However, Entity Framework has a rule that no table names can be the same, regardless of what schema is in place (Mrnka, EF Code first, how to register same table name with different schema?, 2011). This meant that some refactoring was required; at least that problem was addressed (see How Many Projects? below).

After the refactoring was done, some pages did not quite work as expected. This was eventually traced to the views not being fully updated when the C# source files were updated. There is an option to allow the view pages to be compiled right away when a project is built, rather than use dynamic building, but it required unloading the project first (Ugurlu, 2011). While compiling all of the razor views at the start caused the build time to increase, it seemed to result in faster page viewing overall.

The refactoring of the code was not as complete as originally thought; some more cleaning up was required. During a database analysis, it was observed that two tables had multiple columns as foreign keys to the same parent table. The only apparent difference between the two tables was an underscore. As only one column was needed, it took some modifying of the model builder, specifically the Map function, to allow for a single column to be used cleanly (Man, 2012).

The assignment creation form was looked into once more, as there was confusion about an introduced remote validation attribute. The first error was due to a misunderstanding of the remote attribute with additional fields. By not including the

additional field exactly as the property was named, no value was passed in, thus causing an eventual problem when an integer was needed but a nullable integer was expected (Niessner, 2010). Once the routing was fixed, the second error made itself apparent: no error message was printing. Instead, only an asterisk character was showing by the fields. This one took longer to debug, for the problem was actually in the object editor templates that were brought in. Despite claiming to be a decent re-creation of the default razor editor template, the error messages were being suppressed due to a second parameter in the ValidationMessageFor method (Microsoft). Once the second parameter was removed, the attributes and Json calls could use the error messages they were written to output.

At this point, the assignments could be created and deleted with no problems. It was not possible to update the individual answers or sheet names, but points could be assigned to the answers. The professor could also state how each answer should be graded. This was only the first step: the students had to be able to upload their own sheets to allow for the grading to begin.

To allow the students to upload a sheet, they first needed the ability to download a sheet. It is the professor's job to create the template by setting what sheets are initially displayed and what data is to be given to the students. There is already one form that is appropriate to be used for generating templates, so a new mechanism was needed to allow for multiple submit buttons to work at once. In this case, the MultiButtonAttribute was discovered and implemented to allow two or more buttons to be used on a single form and point to different methods in the corresponding controller (Balliauw, 2009).

Setting up the buttons was not enough. At this point, templates needed to be generated from the Excel file. It is fortunate that EPPlus can work with not only files, but streams. Reading a stream was relatively simple: the file just had to be read in as an array

of bytes, and then passed into a MemoryStream object (Källman, How to read a web based file, 2011). Saving an excel package as a stream was also simple: it only needed the SaveAs function with a different stream parameter (Danny, 2011). The reason for two stream objects is due to the first parameter having its stream size fixed upon creation. The other stream was not created by sending a byte array to it, so it is allowed to expand or shrink as required (HeartattacK, 2008). Any sheets that needed to be deleted were done so with a method call, though with using the name of the sheet rather than its position (Källman, EPPlus-Create advanced Excel 2007 spreadsheets on the server, 2011). Once the stream was created, its position needed to be reset before it could be used for downloading purposes (Mischel & chance, 2010). At this point, downloading templates worked as expected: viewing the sheets in Excel confirmed that the data that was to be kept on the sheet stayed on the sheet.

After the initial template data was confirmed working, the adding of students needed an email confirmation check to be sure the proper data was being sent. There were two problems with the email. The first was that the URL data was not loaded correctly. Luckily, there were a number of ways available to get the proper URL data: in this case, it was the domain name that was causing problems (Mitchell, Nazim, & Muñoz, 2008).

The second problem was not due to the email itself, but due to the routing mechanism. A custom route needed to be put in due to the GUID. To further ensure that the right route was chosen, a validation mechanism was put in to make sure that only valid GUIDs got through and the student could be added to the system properly (Hayman, 2007).

Now that students were being taken care of, a minor complaint needed to be addressed. When setting answers, the values and formulas were all grouped by their typed, and then shown in alphabetical order by the cell address. While this could not be confirmed, it's suspected that the table per type hierarchy for some of the database tables was causing the improper ordering scheme for looking over the cells and their values or formulas. While there may have been a way to address this with the proper Linq query, the IComparer method was utilized after the query was executed instead to sort the elements of the resulting list (Microsoft). The advantage of the IComparer was that it could be used in more places than queries. Once the IComparer was implemented, the answers were shown in their intended order.

Attention was shifted back to the template generator temporarily, but for a valid reason. At the time of the original creation, there was no restriction on the username a person would create. Since the username was actually used for part of the filename, this would have to be corrected. Thankfully, the Path class within the System.IO namespace contained functions to check for invalid characters in filenames and directories (Bell, 2010). This was utilized in a new attribute that was placed in the registration forms.

With the template system set up, the student side needed adjusting. To download the generated template for a homework assignment, a form was utilized. The problem that needed addressing here was that certain properties needed to always be sent in the form, but also be not editable. While a new object editor template could have been created, a simpler approach was since discovered. There is a UIHint attribute that can be applied to properties to explicitly state which editor template should be loaded (Wheeler, 2011). Admittedly, this could have been useful prior, but at least the knowledge is now

available for any other properties that need custom work without needing an entire object editor template.

Uploading the assignment as a student to the server was similar to the downloading just mentioned, so nothing special needs to be addressed here. The only thing significant was the new problem that arose from this. How should the sheets be graded, and in what order?

Designing the query that this would entail proved to be very difficult. The goal was to use one query to get information about not only the file, but the specific sheet name as well. As only the latest submitted sheet would be used for grading, a group by query based on the submission date was needed. That would then need to be joined with another table or query to get the related data. It is perhaps fortunate that the query is not executed until the result has to be enumerated: this means the query can be built from separate Linq statements (Le Savard, 2010).

To help visualize the type of query or queries that would get the results needed, a separate program was required this time. Fortunately, LINQPad existed for this purpose. LINQPad is a program that can interface with a number of databases and allow the user to write Linq code to generate the requested results (Albahari, Download LINQPad, 2007). It also has the benefit of showing the SQL equivalent, which is perhaps more familiar to some people.

During the query construction with LINQPad, a database table had to be joined with fields from two different tables. Even though Linq uses a query like syntax, it is not exactly SQL. Joining on multiple fields requires setting two objects, usually anonymous objects, to be equal to each other (Cornell, Skeet, W, & Andersson, 2008). Even still, the

objects must be of the same type or have the same properties in order for the join to work (Warren, 2007). This dictates how the joins and sub queries must be created.

With the help of LINQPad, the needed query was written and put in source control. Attention could be given to the seed function, which populates the database with pre-set data upon database drop and re-creation. The test assignment needed to be put in there to speed up the testing process. The only problem came in with one of the submitted files due to unintentional negligence. When a new DateTime variable is created, it defaults to a date of January 1st, 2001 at midnight. This date is unacceptable for the SQL Server DateTime variable type: the earliest date it accepts is January 1st, 1753 (Orsich, 2011). A missing DateTime variable on object creation was the cause of this error: it was fixed once it was obvious what was going on.

During the query building, alternative methods were looked into to try to simplify the task of grading the sheets. The main item that was looked into was the Workflow technology. Workflows allow programmers to create rules and objects indicating what should happen at each step, all culminating at a terminating point when the work is concluded (Zhu, 2010). Unfortunately, this technology could not be utilized for this project. It was decided in Chapter 4: Technologies Chosen that the .NET Framework version 4 would be used: Workflow requires an update to the 4.0.1 version of .NET, which was not stated explicitly as a version that could be used or upgraded to (Danielson, 2011).

Once the decision was made to use the existing technologies, the coding was (conceptually) simple. The order of how to grade the items was known, so looping through was possible. The tables were coded appropriately. It was fortunate that no foreign key fields were shared across the derived tables of CellBase; otherwise, extra key

mappings could have resulted due to an explicit feature of the Entity Framework (Mrnka, EF Code-First table per type with base class DbSet, 2011). It was also fortunate that the derived types could have their own DbSet variables on hand to make table joining easier (archil, 2012).

After some adjustments with database errors and off by one errors, the assignments were being graded appropriately. Any unrecognized answers that were supposed to be exact allowed for choosing what was right or wrong once: it remembered the decision for all future attempts on that question, even re-grades. Other question types were simply right or wrong based on expected data. There is no clean way to provide half credit for cells that should be integers but actually are not at this time.

With the grading of assignments working, the next task to handle was the exporting of grades. As the AGLS is purely meant for grading sheets, there needs to be a way to bring the data to other formats. For this project, three formats were focused on: CSV, JSON, and XML.

Writing the actual query took some time in LINQPad. The best way at the time to build the needed query was to do it in pieces. A special technology called lazy evaluation helps with this process: it ensures that the execution of the SQL statement and retrieving of rows is handled as late as possible (wesdyer, 2007). By doing it in pieces, a single query statement is built instead of multiple statements that call the database multiple times.

The original plan for the pieces was to attempt a “not in” query: that is, select all students that did not have a grade, and manually give a grade of 0%. Not only was this plan too cumbersome to even try (it cannot be explained why this thought process came to the forefront), but the resulting query would not have used not in at all. Instead, the

resulting query would have become a “not exists” query, which usually gets the same results in a faster time (Russo, 2008).

This concept of SQL query expansion was further explored during the final solution of the method that gets all grades for the term. Instead of the not in query above, a left join query was attempted for construction. A cross join between the students and assignments was required: thankfully, it was simple to set up (Morgan, 2008). The cross join result was then left joined with the known grades for each combination. Any combinations left out were given a 0% grade. The end result upon evaluation was not a left join at all. Instead, it used the SQL Server exclusive apply operator instead.

The apply operator is often used to “join a table to a table-valued function so the function is evoked for each row returned from the table.” (Sheldon, 2012) That definition is strange, for no table-valued functions were seen in the generated result. A manual left join query as written in Management Studio to confirm that the same results could be gotten; that query was then further analyzed to determine if there were any inefficiencies with the left join query versus the apply query. While the apply version had some more operations, the time spent on various indexes was roughly the same. This was confirmed independently by Arshad Ali: for what is generally a simple left join query, there is virtually no difference in execution times; just execution plans change (Ali, 2010). It was decided that this would not be something to worry about for this project.

What did need focusing, however, was a runtime exception that was taking place to get the grades. Unfortunately, LINQPad is not perfect: it allows some statements that would not work in Visual Studio’s C# compiler. By performing a left join and checking if the entire joined object was null, a runtime error was the result. Specifically, it states that “The argument to DbIsNullExpression must refer to a primitive or reference type”.

(bommob1, 2011). Fortunately, a solution was found quickly: instead of checking if the entire object is null, check just one property of the object. That fixed up getting the grades, which meant the challenge now was exporting them.

Writing the XML solution was confusing, but only because of the multitude of options. One of the original plans pursued was to write a derived ContentResult to return the XML document, but that method worked best for sending XML to a view and not as a file to be downloaded (Anand, 2012). The next idea was to convert the XML to a string using a StringWriter to send the data appropriately (Kean, 2006). While this worked, the data would eventually have to be converted back into a stream to work with the FileStreamResult. While there was sample code to convert strings to streams and vice~versa, such an application would prove to be more useful for the JSON and CSV formats (Martin, 2009). As such, the decision was made to just serialize the XML, and convert the serialized data to the appropriate MemoryStream (Microsoft, 2011). The serialization had a runtime failure early on, however, due to the object creation. The original object used dictionaries to represent the data. There is no consistent answer for why dictionaries cannot be serialized, though a number of workarounds have been proposed (serialhobbyist, 2009). Ultimately, the object that held the grade data was re-written to use plain properties instead of dictionaries.

Writing the JSON solution was simpler in the long run. Similar to the XML solution above, the .NET Framework provided a way to serialize objects into JSON. Unfortunately, the serialization method only returned a string for its output (Dayan, 2009). Once the string was gotten, however, it could be converted to a stream as shown above.

The CSV file proved to be the trickiest conceptually. While the other two formats allowed for serializing an object in a generic manner, CSV files usually have to be handled in a different way. In this case, the first line of header data was written with a separate query, and then the grades were all analyzed one at a time and placed in an ever growing StringBuilder. Once completed, the resulting string was Memory Streamed.

With the grades working, another required task was taken care of relatively quickly. Similar to the original AGLS, all templates that are generated contain a very hidden worksheet with a way to identify the student that is supposed to download the sheet. When it comes time to grade the sheet, the code looks for the identifying information on the sheet to confirm a match. If the identifying information is not found, the sheet is flagged and the student receives a -100% for the entire submission. As the cheating possibility is only detected after each request to decide right or wrong answers, the professor can objectively decide what is truly correct or incorrect without influence.

Of course, there are some assignments that the professor already knows what should happen for each situation regardless. An option was added to the assignment creation form to allow for automatic grading. This turns the AGLS into an interactive-less web service: by giving up the grading controls, students can upload an Excel spreadsheet and get instant results. This can allow the professor to focus on other tasks instead of worrying about grading.

At this point, the required items for this AGLS are now in the system. There are no plans to add anything major to the AGLS. Minor enhancements and unexpected bug fixes can still take place, however.

Writing Unit Tests

As stated, one of the goals of this project was to introduce the idea of unit testing. The goal of the unit tests is to test a few individual pieces such that those will work appropriately for when it comes time to develop the application.

Three sets of unit tests were written. The first one was for the Person model. These tests were written to ensure the integrity of the planned database model, as well as confirm that public readonly properties would work regardless of how the data is stored. A model in this context is only an object: it does not matter how the objects are collected. The unit tests here showed the integrity of the single model, allowing for more confident code later on.

The next set of tests expanded on the Person test by testing the Person Repository. The point of repositories is to enable abstracted access to the underlying collection objects: in this case, the Persons (or People if you prefer). As it's not always convenient to set up a separate database, this was the first instance of using Moq objects. Moq is a library that allows the user to mock an object: that is, pretend that it is the intended object, only with specifically coded substitutions on actions as required. It can avoid the problems of having an entire object that is merely duplicated with minor changes: this way, the core object can stay the same and Moq can be configured to perform different actions as required (Williams, 2009).

The last set of tests done was for email. The Postal library already had some tests written for it in its source code, but the idea for the tests that needed to be written was that an email could be "sent" without actually sending one. Again, Moq came into play to capture the email that would have been sent normally into a variable so that it could be

analyzed. It is unknown if this test batch was needed as there were already tests, but no documentation was found on whether or not testing an already tested library was viable.

There is a caveat to these unit tests, however. These were written in the infancy of the project. There is a chance that the code these tests targeted changed so much that the tests are no longer valid. This was not investigated further.

Making Tough Decisions

Even with the testing and planning that took place, a number of choices had to be made throughout the whole process. The shape of the AGLS project was dictated by these choices.

Storing Excel Files

One of the early decisions that had to be made was how to store the Excel data that came in. There was a spirited discussion about whether to try to parse all of the data in one go, or only parse what is needed. The advantage of the former is that the parsing is done right away, with no apparent need to redo it. The advantage of the latter is that not all of the data is in the database. Depending on how many users wanted to use the service, it could easily be flooded with many rows just about…well, Excel rows. Well, more like Excel cells. Ultimately, the idea of storing just what is needed and parsing the rest on demand won out.

Date Time Pickers

Some of the models need not only a date, but a time to go with the date as well. While jQuery UI already offers a date picker that works with HTML input elements cleanly, it does not deal with time (jQuery Foundation and the jQuery UI Team, 2012). This required searching for a new plugin that deals with time, and preferably also the date as well.

The first one found was Trent Richardson's datetimepicker add-on. It effectively extends the normal datepicker by adding time information that can be adjusted by the user. All of the date functionality is still present (Richardson, 2010). The fact that the library is compatible with the MIT license for commercial purposes is a plus. The only disadvantage was the inability to use the default jQuery UI CSS file that Microsoft used: it caused problems with the sliders during testing.

The second one identified was the dynDateTime library by someone who goes by "thetoolman". It is an old plugin that was forked from a DHTML Calendar plugin to try to add a date and time picker (thetoolman, 2008). This library is an old one that was apparently made in jQuery's infancy: trying to run the library caused errors in trying to even load the package. Since dynDateTime could not work consistently, it was discarded.

The last one found was the Any+Time™ DatePicker/TimePicker AJAX Calendar Widget. The TM is necessary, as this is a professional product that offers multiple options for dates and times with an allegedly simple script (Andrews III, 2012). Its licensing scheme is a Creative Commons variation that forbids commercial use: explicitly, it states that it becomes a violation of someone has to pay to access the plugin. As one of the future goals of this AGLS is to market it for commercial use, this prevents the use of Any+Time™.

Reading and Storing Excel Data Temporarily

This goes hand in hand with the Storing Excel Files section above. Whenever an Excel file is opened, its data is available for reading and writing. When the file is closed, the file's data is no longer available. As having access to that data at any time is important, there was the question of not only when to read the files, but whether to store the data in an intermediate data structure.

The reading and writing of files generally takes place in a model or some equivalent in the MVC approach. The view is generally not where file reading or writing should take place: the view is for displaying data. This does not mean that there cannot be any logic in the view: it's generally not the right place.

The idea of an intermediate data structure was enticing for awhile, but the plan was discarded relatively quickly. While this is conjecture, it probably took Jan Källman a good while in order to develop his EPPlus library. Even with that as a reference to make either a simple view model or even a spreadsheet agnostic data structure, it would take longer to code and test than there is time to develop the AGLS.

Should more file formats be introduced, having common objects that can be converted could be nice. However, ultimtely it comes down to what works. To quote a colleage on a different project: "Make it work, make it better, make it fast." (Cannon, 2012) For now, the reading of Excel files will take place in the views as required.

How Many Projects?

When a website using the MVC framework is made using a non empty template, there is an option to add a unit testing project alongside it. This means two projects can be created by default. While the empty template was used to start the AGLS, four projects was always the goal: one for the core application, one for the database models and repositories, one for unit testing, and one for everything else, dubbed utilities.

This eventually caused problems when some items in utilities needed to interact with the database models and vice~versa. C# libraries and projects cannot be referenced circularly: otherwise a deadlock would result with the compiler being unable to build the libraries (Smith, 2008). The unit test project could be factored out at this point: no matter

what, no other project would reference it. The question was thus focused on the three other projects: how should the files and projects become organized?

What is perhaps funny is that instead of combining everything into a big project as was advised, even more projects were created to handle specific tasks. Not counting the unit test project, there are twelve projects in use at the time of writing. The database project was split into a model/table project and the current database project, now repurposed strictly for accessing the tables. The utilities project was split into multiple projects: most of the subfolders became subprojects of their own, with each focusing on the folder's underlying purpose or type. The only project that needed some slightly special handling was the Excel project: it needed items from two other projects that would have caused their own circular reference if it was not included.

Entity Framework: What's First?

The Entity Framework is an object relational framework that allows mapping database tables to plain old code objects. What makes Entity Framework unique is that there are three ways to allow mapping the tables to objects and vice~versa.

While Appendix B can go into more detail on the three methods, the one chosen for this project was Code First. One of the major benefits of code first was having an easy method to populate the database either every time the application started or when the model is updated based on new changes that had to be made. This way, there was no need to worry about the data that existed in the database and the website would not always have to be run with the same steps just to get the same data in the database manually.

While Database First could have allowed for absolute database control and allowed for custom things such as stored procedures and views, Code First is database agnostic. This means that code first will work no matter what database system the AGLS

eventually goes to in production. As one of the stated goals was to eventually allow for multiple database systems to be used with the AGLS, it was important to keep that in mind the whole way through the design.

Database Table Inheritance

There are three characteristics of object-oriented programming that make it what it is: encapsulation, polymorphism, and inheritance (Microsoft, 2010). The focus here is inheritance, which allows for extending the functionality of classes or implementing interfaces.

Relational database tables in general do not deal with inheritance: they are not objects (ChssPly76, 2009). To map inherited objects to database tables requires specific planning and knowledge of the framework in question. As the Entity Framework is in use, there are three methods to look into.

The first method, which is the default method for Entity Framework, is table per hierarchy, or TPH. With TPH, the base object that is inherited becomes a table, with all other children sharing the same table in the database. An extra column, a discriminator, is then added to help determine which object the row really represents (Manavi, 2010). By having the base object and its inherited children share the same database table, lookups are theoretically faster since there is no database joining. However, this represents a denormalized schema, which is not always desired by database admins.

The second method is known as table per type, or TPT. With TPT, the base object and its inherited children are all given individual tables to use; the database relies on foreign key relationships to join the tables appropriately (Manavi, Inheritance with EF Code First: Part 2 – Table per Type (TPT), 2010). To get the data of a specific object, at least one extra join is required to get to the proper table. While the database is

normalized, it is trickier in code to write the relationships, as there is not supposed to be a foreign key property. This makes deleting a parent object trickier: Entity Framework does not automatically perform a cascade delete operation on table per type tables and objects, and thus a manual way is required (Slauma, Problems using TPT (Table Per Type) in EF 4.2 and deletion of parent objects, 2011).

The last method of table inheritance mapping is known as table per concrete type, or TPC. With TPC, the base object is not represented as a table in the database: instead, the children are by mapping the inherited properties to the children tables (Manavi, Inheritance with EF Code First: Part 3 – Table per Concrete Type (TPC), 2011). It is hard to say how useful this pattern is, however. TPC requires a number of configurations to make it work well. One of those configurations deals with key generation. TPC tables that have the same base object cannot have the same primary key values across all of the children. Pretend that an identify column was split between two or tables. That is what would have to take place. Since identity/auto incrementing columns are only tied to one table and not multiple, a manual key generating method is required.

TPC was immediately written off: based on the research, it seemed to have too many issues for not enough gain. The decision was between TPH and TPT. Entity Framework handles TPH automatically, and TPT requires some configuration to make it work. The issues of database normalization had to be weighed in to determine if such a goal was still viable.

The goal was to try TPT to keep database normalization alive at first, and then switch to TPH if further problems developed. The only tricky part was in the deleting of related objects. There is an assignment table that contains answer worksheets, which contain cell bases. At the time of writing, a cell base is effectively an abstract object

which is inherited by a cell value table and cell formula table. When it came to deleting, the normal way of deleting an assignment would not work due to the mapping of the tables. An overriding delete method was written to address this issue. After a few tries with deleting the rows the wrong way, the assignment and its related objects were able to be removed from the database, and no errors resulted from the deleting.

Chapter 6: Conclusions and Future Work

The journey from planning the AGLS to putting the system together has been a long, but rewarding journey. Early on, it was known that rebuilding and re-visioning this system to utilize current best practices, tools, and methodologies would be very complex. The original system had to be studied to get a general idea of how the old AGLS performed its duties. Many professors had to be interviewed to identify both the great parts and the areas of improvement with the system. The various file formats had to be studied to determine how feasible it would be to implement them should hand coding be required.

A number of test programs not related to the AGLS were written to ensure that the programming could be done for the colossal project. Libraries and frameworks were identified to make the programming easier to work with. Even unit testing and version control was brought in to ensure a relatively clean code base so that future programmers could work on this system if they so choose to.

A special emphasis must be made on the libraries and frameworks. Those two technologies helped speed the process of developing the code in a timely manner. Details such as how to parse an Excel spreadsheet or send an email asynchronously did not have to be worried about. Instead, the main details to focus on included what data to grab from an Excel sheet, or how to compose the email to be sent to a professor or student. Even when specialized code had to be written, the tools made the job that much easier.

The role of a programmer is changing due to these libraries and frameworks. It is getting to the point where programmers shouldn't have to re-create data structures or libraries just to perform a task over and over again. The important lesson to be learned is

that libraries and frameworks are around to help programmers with creating a new project or re-visioning an old one. Time should be spent more on coding the unique items of a project rather than the entire package if it can be helped.

While there is always more that can be done with the system (see Appendix D for goals on moving this new AGLS to a production ready environment), the work done should demonstrate one way of how to re-vision an old system. With the re-visioning done here, it should be easier for professors to create and grade assignments, and students to submit assignments and get graded on them. The AGLS has been shown, and proven, that it can assist everyone within the university spectrum: hopefully it can be utilized everywhere in the future.

References

.net - Linq over Stored Procedures. (2009). Retrieved March 2012, from Stack Overflow: http://stackoverflow.com/questions/726908/linq-over-stored-procedures

File: SASS_REFERENCE. (2012, August 20). Retrieved August 22, 2012, from Sass - Syntactically Awesome StyleSheets: http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html

Adamyan, A. (2010, July 29). *Addition to ASP.NET MVC Localization - Using routing.* Retrieved August 31, 2012, from While my keyboard gently weeps...: http://adamyan.blogspot.com/2010/07/addition-to-aspnet-mvc-localization.html

Adamyan, A. (2010, February 22). *ASP.NET MVC 2 Localization complete guide.* Retrieved August 29, 2012, from While my keyboard gently weeps...: http://adamyan.blogspot.com/2010/02/aspnet-mvc-2-localization-complete.html

Albahari, J. (2007). *Download LINQPad.* Retrieved September 28, 2012, from LinqPAD: http://www.linqpad.net/

Albahari, J. (n.d.). *C# Concepts: Value vs Reference Types*. Retrieved September 10, 2012, from Joseph Albahaari: http://www.albahari.com/valuevsreftypes.aspx

Ali, A. (2010, March 9). *SQL Server CROSS APPLY and OUTER APPLY.* Retrieved October 6, 2012, from MSSQLTips: http://www.mssqltips.com/sqlservertip/1958/sql-server-cross-apply-and-outer-apply/

Allen, B. (2009, October 20). *JSON Problem - JsonRequestBehavior to AllowGet.* Retrieved September 7, 2012, from The Official Microsoft ASP.NET Forums: http://forums.asp.net/post/3467877.aspx

Allen, S. (2010, March). *Model Validation & Metadata in ASP.NET MVC 2.* Retrieved September 3, 2012, from Extreme ASP.NET: http://msdn.microsoft.com/en-us/magazine/ee336030.aspx

amit_g. (2011, August 16). *MVC3 Html.EditorFor isn't rendering a single thing in my View*. Retrieved September 13, 2012, from StackOverflow: http://stackoverflow.com/a/7085446/445373

Anand, V. (2012, June 22). *In ASP.NET MVC, How to return an xml document to a View.* Retrieved October 5, 2012, from StackOverflow: http://stackoverflow.com/a/11159801/445373

Andrews III, A. M. (2012). *Any+Time™ DatePicker/TimePicker AJAX Calendar Widget.* Retrieved September 8, 2012, from HTML5, AJAX, JSON and XML Consulting and Training by Andrew M. Andrews III (SM): http://www.ama3.com/anytime/

andyr. (2012, January 5). *StyleCop*. Retrieved September 10, 2012, from CodePlex: http://stylecop.codeplex.com/

archil. (2012, June 10). *Entity Framework 4 and MVC 3 - Inheritence and Loading Derived Entities.* Retrieved October 3, 2012, from StackOverflow: http://stackoverflow.com/a/10968684/445373

Babcock, C. (2009, January 2). *Open Document Format Has Been Accepted By 16 Governments.* Retrieved March 2012, from InformationWeek: http://www.informationweek.com/software/productivity-applications/open-document-format-has-been-accepted-b/212700444

Balliauw, M. (2009, November 26). *Supporting multiple submit buttons on an ASP.NET MVC view.* Retrieved September 25, 2012, from Maarten Balliauw {blog}:

http://blog.maartenballiauw.be/post/2009/11/26/Supporting-multiple-submit-buttons-on-an-ASPNET-MVC-view.aspx

Bell, J. (2010, March 12). *.Net: How do I check for illegal characters in a path?* Retrieved September 27, 2012, from StackOverflow: http://stackoverflow.com/a/2435941/445373

bommob1. (2011, June 7). *Error: The argument to DbIsNullExpression must refer to a primitive or reference type* . Retrieved October 5, 2012, from msdn forums: http://social.msdn.microsoft.com/Forums/eu/adodotnetentityframework/thread/a5d0b69e-f3b0-48dc-be67-dadea1766c68

Brind, M. (2012, June 2). *ASP.NET MVC Uploading and Downloading Files.* Retrieved September 5, 2012, from mikesdotnetting: http://www.mikesdotnetting.com/Article/125/ASP.NET-MVC-Uploading-and-Downloading-Files

Burbeck, S. (1992). *How to Use Model-View-Controller (MVC)*. Retrieved August 29, 2012, from http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

Burnett, B. (2011, March 25). *Correcting MVC 3 EditorFor Template Field Names When Using Collections*. Retrieved September 14, 2012, from Brant Burnett's Development Blog: http://btburnett.com/2011/03/correcting-mvc-3-editorfor-template-field-names-when-using-collections.html

Cannon, M. (2012, June). Efficiency vs. Reliability for StepMania 5. (J. Felds, Interviewer)

ChssPly76. (2009, October 14). *How to do Inheritance Modeling in Relational Databases?* Retrieved September 23, 2012, from StackOverflow: http://stackoverflow.com/a/1567968/445373

Cornell, J., Skeet, J., W, N., & Andersson, K. (2008, December 17). *How to do joins in LINQ on multiple fields in single join.* Retrieved September 28, 2012, from StackOverflow: http://stackoverflow.com/questions/373541/how-to-do-joins-in-linq-on-multiple-fields-in-single-join

counsellorben. (2011, August 24). *Nested edit templates in mvc razor*. Retrieved September 14, 2012, from StackOverflow: http://stackoverflow.com/a/7180769/445373

Creative Commons. (n.d.). *Creative Commons*. Retrieved September 8, 2012, from Creative Commons: http://creativecommons.org

crocgod. (n.d.). *Tidy.NET*. Retrieved March 2012, from Free Development software downloads at SouceForget.net: http://souceforge.net/projects/tidynet/

Danielson, S. (2011, April 18). *Microsoft .NET Framework 4 Platform Update 1.* Retrieved September 29, 2012, from The .NET Endpoint: http://blogs.msdn.com/b/endpoint/archive/2011/04/18/microsoft-net-framework-4-platform-update-1.aspx

Danny. (2011, June 23). *Using EPPlus with a MemoryStream.* Retrieved September 25, 2012, from StackOverflow: http://stackoverflow.com/a/6460978/445373

DarthObiwan. (n.d.). *Html AGility Pack*. Retrieved March 2012, from Codeplex: http://htmlagilitypack.codeplex.com

databasedev.co.uk. (n.d.). *Export Access 2007 Data to XML Format | Database Solutions for Microsoft Access*. Retrieved August 31, 2012, from databasedev.co.uk: http://www.databasedev.co.uk/access2007_export_xml.html

Dayan, P. (2009, March 12). *Convert objects to JSON in C# using JavaScriptSerializer.* Retrieved October 5, 2012, from Pini Dayan: http://blogs.microsoft.co.il/blogs/pini_dayan/archive/2009/03/12/convert-objects-to-json-in-c-using-javascriptserializer.aspx

Deckmyn, D. (2006, June 23). *Belgian government chooses OpenDocument.* Retrieved March 2012, from ZDNet: http://www.zdnet.co.uk/news/desktop-apps/2006/06/23/belgian-government-chooses-opendocument-39276978/

Dighe, R. (2005, August 11). *Disabling browser's back functionality on sign out from Asp.Net.* Retrieved September 5, 2012, from The Code Project: http://www.codeproject.com/Articles/11225/Disabling-browser-s-back-functionality-on-sign-out

Dimitrov, D. (2011, June 17). *How to validate uploaded file in ASP.NET MVC?* Retrieved September 5, 2012, from StackOverflow: http://stackoverflow.com/a/6388927/445373

Duffy, W. (2009, October 9). *CSS class property in ASP.NET MVC HtmlAttributes* . Retrieved September 15, 2012, from William Duffy: http://www.wduffy.co.uk/blog/css-class-property-asp-net-mvc-htmlattributes/

Dykstra, T. (2011, April 11). *Creating an Entity Framework Data Model for an ASP.NET MVC Application (1 of 10).* Retrieved August 2012, from Official Microsoft Site: http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

ECMA International. (2006, June). ECMA-334. *C# Language Specification*(4th Edition). Geneva, Switzerland: Microsoft. Retrieved August 29, 2012, from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Elliott, D. (2010, Januar 4). *ASP.Net MVC2: ModelState is invalid, but I don't know why*. Retrieved September 17, 2012, from StackOverflow: http://stackoverflow.com/a/2001437/445373

Esposito, D. (2010, June 21). *ASP.NET MVC and File Uploads.* Retrieved September 5, 2012, from dotnetslackers.com: http://dotnetslackers.com/articles/aspnet/ASP-NET-MVC-and-File-Uploads.aspx

Extreme Experts. (2007). *ViewState in ASP.NET*. Retrieved August 29, 2012, from Extreme Experts: http://www.extremeexperts.com/Net/Articles/ViewState.aspx

Ferner, D. M., & Patterson, D. L. (2012, February). Parsing Web Sites. (J. Felds, Interviewer)

Firefox. (2012). *Warning Unresponsive script - What it means and how to fix it*. Retrieved September 15, 2012, from Firefox Help: http://support.mozilla.org/en-US/kb/warning-unresponsive-script

Foundation for IT Education. (2011). *Our Mission*. Retrieved August 30, 2012, from Foundation for IT Education: http://www.edfoundation.org/mission.htm

Fowler, M. (2004, January 23). *Inversion of Control Containers and the Dependency Injection pattern.* Retrieved September 1, 2012, from Martin Fowloer: http://www.martinfowler.com/articles/injection.html

Free Software Foundation, Inc. (2007, June 29). *The GNU General Public License v3.0.* Retrieved August 31, 2012, from http://www.gnu.org/copyleft/gpl.html

Free Software Foundation, Inc. (2007, June 29). *The GNU Lesser General Public License v3.0.* Retrieved August 31, 2012, from http://www.gnu.org/licenses/lgpl.html

Galloway, J. (2011, April 21). *MVC Music Store.* Retrieved September 2, 2012, from Official Microsoft Site: http://www.asp.net/mvc/tutorials/mvc-music-store

Gateway. (n.d.). *Gateway Foundation*. Retrieved August 30, 2012, from Gateway: http://us.gateway.com/gw/en/US/content/gateway-foundation

Gillum, C. (2008, August 26). *LINQ-to-SQL vs stored procedures?* Retrieved March 2012, from Stack Overflow: http://stackoverflow.com/questions/14530/linq-to-sql-vs-stored-procedures

Gloor, R. (2012, February 8). *ASP.NET Provider dependency injection with Ninject 3.0.0.* Retrieved August 28, 2012, from planetgeek.ch: http://www.planetgeek.ch/2012/02/08/asp-net-provider-injection-with-ninject-3-0-0/

GNU. (1996). Retrieved August 31, 2012, from The GNU Operating System: http://www.gnu.org/

Goldstone, S. (2011, March 10). *How do you create a dropdownlist from an enum in ASP.NET MVC?* Retrieved September 18, 2012, from StackOverflow: http://stackoverflow.com/a/5255108/445373

Gu, S. (2007, August 16). *LINQ to SQL (Part 6 - Retrieving Data Using Stored Procedures)*. Retrieved August 31, 2012, from ScottGu's Blog: http://weblogs.asp.net/scottgu/archive/2007/08/16/linq-to-sql-part-6-retrieving-data-using-stored-procedures.aspx

Gu, S. (2010, July 2). *Introducing "Razor" – a new view engine for ASP.NET*. Retrieved March 2012, from ScottGu's Blog: http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx

Haack, P. (2008, October 23). *Model Binding To A List*. Retrieved September 15, 2012, from you've been hacked and you like it: http://haacked.com/archive/2008/10/23/model-binding-to-a-list.aspx

Hare, A., & ldsenow. (2009, August 6). *How to set a constant decimal value*. Retrieved September 15, 2012, from StackOverflow: http://stackoverflow.com/questions/1236402/how-to-set-a-constant-decimal-value

Harman, D. (2011, June 23). *ASP.Net MVC 3 Custom Membership Provider with Repository Injection.* Retrieved August 28, 2012, from Dan Harman: http://www.danharman.net/2011/06/23/asp-net-mvc-3-custom-membership-provider-with-repository-injection/

Hayman, P. (2007, July). *IsGuid() (Regular Expression Guid Match)* . Retrieved September 26, 2012, from GeekZilla: http://www.geekzilla.co.uk/View8AD536EF-BC0D-427F-9F15-3A1BC663848E.htm

Headcrash Industries. (2011, September 14). *Custom Display and Editor Templates with ASP.NET MVC 3 Razor*. Retrieved September 14, 2012, from headcrash industries: http://www.headcrash.us/blog/2011/09/custom-display-and-editor-templates-with-asp-net-mvc-3-razor/

HeartattacK. (2008, August 25). *Memorystream Not Expandable: Invalid Operation Exception.* Retrieved September 25, 2012, from HeartattacK: http://weblogs.asp.net/ashicmahtab/archive/2008/08/25/memorystream-not-expandable-invalid-operation-exception.aspx

Hillenius, G. (2012, January 04). *Open document standards mandatory in Hungary government.* Retrieved March 2012, from Joinup:

https://joinup.ec.europa.eu/news/open-document-standards-mandatory-hungary-government

Hollhumer, R. (2012, January 19). *Operation is not valid due to the current state of the object.* Retrieved September 15, 2012, from Renso Hollhumer: http://geekswithblogs.net/renso/archive/2012/01/19/operation-is-not-valid-due-to-the-current-state-of.aspx

Horst, B. (2008, January 8). *Converting SQL to LINQ, Part 7: UNION, TOP, Subqueries (Bill Horst)*. Retrieved September 17, 2012, from MSDN Blogs: http://blogs.msdn.com/b/vbteam/archive/2008/01/08/converting-sql-to-linq-part-7-union-top-subqueries-bill-horst.aspx

IBM. (2006, October 27). *DB2 Database for Linux, UNIX, and Windows*. Retrieved August 30, 2012, from http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/c0004100.htm

Independentsoft. (2012). *ODF .NET*. Retrieved September 8, 2012, from Independentsoft: http://www.independentsoft.de/odf/

Independentsoft. (2012). *Purchase*. Retrieved September 8, 2012, from Independentsoft: http://www.independentsoft.de/purchase.html

Jacobson, N. (2001, November 4). A Method for Normalizing Students' Scores When Employing Multiple Graders. *33*, pp. 35-38.

James, T. (2012, March 12). *jQuery and jQuery UI Fallbacks*. Retrieved September 8, 2012, from timjames.me: http://timjames.me/jquery-and-jquery-ui-fallbacks

jgauffin. (2011, November 16). *best way to multi-language asp.net mvc 3.* Retrieved August 31, 2012, from Stack Overflow: http://stackoverflow.com/questions/8151114/best-way-to-multi-language-asp-net-mvc-3

jminatel. (2010, January 19). *Testing ASP.NET WebForms*. Retrieved March 2012, from Wrox P2P: http://p2p.wrox.com/content/articles/testing-aspnet-webforms

Jones, D. (2010, April 26). *ASP.NET MVC 2 Templates*. Retrieved September 13, 2012, from DalSoft: www.dalsoft.co.uk/blog/index.php/2010/04/26/mvc-2-templates/

jQuery Foundation and the jQuery UI Team. (2012). *Datepicker Demos & Demonstration.* Retrieved September 8, 2012, from jQuery UI: http://jqueryui.com/demos/datepicker/

Kaimal, R. (2010, May 11). *Running ASP.NET Webforms and ASP.NET MVC side by side*. Retrieved August 29, 2012, from Raj Kaimal: http://weblogs.asp.net/rajbk/archive/2010/05/11/running-asp-net-webforms-and-asp-net-mvc-side-by-side.aspx

Källman, J. (2009, November). *EPPlus-Create advanced Excel 2007spreadsheets on the server.* Retrieved August 2012, from Codeplex: http://epplus.codeplex.com/

Källman, J. (2011, January 9). *EPPlus-Create advanced Excel 2007 spreadsheets on the server.* Retrieved September 25, 2012, from Deleting work sheet if it already exists in the workbook: https://epplus.codeplex.com/discussions/240635

Källman, J. (2011, May 3). *How to read a web based file.* Retrieved September 25, 2012, from EPPlus-Create advanced Excel 2007 spreadsheets on the server: http://epplus.codeplex.com/discussions/256064

Kamyar. (2011, October 31). *Foreign Keys in Entity Framework - Cycles or multiple cascade paths error.* Retrieved August 28, 2012, from StackOverflow: http://stackoverflow.com/a/7951150/445373

Kay, D. (1998). Large Introductory Computer Science Classes: Strategies for Effective Course Management. *SIGCSE '98*, (pp. 131-134). Atlanta.

Kay, D. G. (n.d.). *Managing Large Introductory CS Courses*. Retrieved April 2012, from http://www.ics.uci.edu/~kay/pubs/managing-large-courses.html

Kean, D. M. (2006, May 21). *Serialize object to string.* Retrieved October 5, 2012, from http://social.msdn.microsoft.com/Forums/en-US/csharplanguage/thread/5d08bc28-5b61-4c5a-8c4b-4665b1c929ea/

Khan, S. (2010, December 19). *http://www.shahnawazk.com/2010/12/dependency-injection-in-aspnet-mvc-3.html.* Retrieved August 24, 2012, from Shahnawaz Khan's Blog: http://www.shahnawazk.com/2010/12/dependency-injection-in-aspnet-mvc-3.html

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., et al. (1997). Aspect-Oriented Programming. *Proceedings of the European Conference on Object-Oriented Programming. 1241*, pp. 220-242. Springer-Verlag.

Kline, D. D. (2012, February). The Many Actions for a Single Answer. (J. Felds, Interviewer)

Kline, D. D., Ferner, D. M., & Patterson, D. L. (2012, February). More Parsing Requests. (J. Felds, Interviewer)

Kondo, T. (2010, November 1). *ASP.NET MVC Model Binding IList in an Editor Template*. Retrieved September 14, 2012, from StackOverflow: http://stackoverflow.com/a/4069396/445373

Korpela, J. (2003, September 28). *Methods GET and POST in HTML forms - what's the difference?* . Retrieved September 11, 2012, from http://www.cs.tut.fi/~jkorpela/forms/methods.html

Krysmanski, S. (2012, January 25). *IDisposable, Finalizer, and SuppressFinalize in C# and C++/CLI*. Retrieved September 10, 2012, from CodeProject: http://www.codeproject.com/Articles/319826/IDisposable-Finalizer-and-SuppressFinalize-in-Csha

Lagadec, P. (2006). OpenOffice / OpenDocument and MS Office 2007 / Open XML security. *PacSec 2006*, (p. 25).

Le Savard, D. (2010, February 22). *Using max and join together in linq using c#.* Retrieved September 28, 2012, from StackOverflow: http://stackoverflow.com/a/2311318/445373

Leeks, S. (2011, April 13). *ASP.NET MVC: Partial rendering and AjaxAttribute.* Retrieved September 2, 2012, from MSDN Blogs: http://blogs.msdn.com/b/stuartleeks/archive/2011/04/13/asp-net-mvc-partial-rendering-and-ajaxattribute.aspx

Levi. (2009, June 25). *ASP.NET MVC ambiguous action methods*. Retrieved September 18, 2012, from StackOverflow: http://stackoverflow.com/a/1045616/445373

Lister, J. (2009, February 4). *Logging in without a password – certificates & ssh*. Retrieved September 8, 2012, from Jay by Jay Fresh: http://jaybyjayfresh.com/2009/02/04/logging-in-without-a-password-certificates-ssh/

ŁukaszW.pl. (2010, August 9). *ASP.NET MVC redirect from attribute*. Retrieved September 13, 2012, from StackOverflow: http://stackoverflow.com/a/3439121/445373

Man, M. (2012, May 27). *Entity Framework : Fluent API makes duplicate foreign keys in one to zero or one*. Retrieved September 20, 2012, from StackOverflow: http://stackoverflow.com/a/10775768/445373

Manavi, M. (2010, December 24). *Inheritance with EF Code First: Part 1 – Table per Hierarchy (TPH)*. Retrieved September 20, 2012, from Enterprise .Net: http://weblogs.asp.net/manavi/archive/2010/12/24/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-1-table-per-hierarchy-tph.aspx

Manavi, M. (2010, December 28). *Inheritance with EF Code First: Part 2 – Table per Type (TPT)*. Retrieved September 20, 2012, from Enterprise .Net: http://weblogs.asp.net/manavi/archive/2010/12/28/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-2-table-per-type-tpt.aspx

Manavi, M. (2011, January 3). *Inheritance with EF Code First: Part 3 – Table per Concrete Type (TPC)*. Retrieved September 20, 2012, from Enterprise .Net: http://weblogs.asp.net/manavi/archive/2011/01/03/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-3-table-per-concrete-type-tpc-and-choosing-strategy-guidelines.aspx

Martin, T. (2009, February 19). *C# Convert String to Stream, and Stream to String.* Retrieved October 5, 2012, from C# 411: http://www.csharp411.com/c-convert-string-to-stream-and-stream-to-string/

Matthews, K. (2008). *Development and Evaluation of an Adaptive Grading/Learning System (AGLS).* Masters Thesis, University of North Carolina Wilmington, Computer Science, Information Systems and Operations Management, Wilmington.

Matthews, K. (2008). *Development of an Adaptive Grading/Learning System (AGLS).* Wilmington.

McKenzie, C. (2010, August 11). *How do I get the collection of Model State Errors in ASP.NET MVC?* Retrieved September 17, 2012, from StackOverflow: http://stackoverflow.com/a/3459569/445373

Meacham, S. (2009, December 17). *Making Entity Framework (v1) work, Part 1: DataContext lifetime management.* Retrieved September 3, 2012, from Sam's Code: http://samscode.com/index.php/2009/12/making-entity-framework-v1-work-part-1-datacontext-lifetime-management/

meisol. (2010, December 28). *Asp.Net Mvc Hidden Field from Data Annotations*. Retrieved September 12, 2012, from StackOverflow: http://stackoverflow.com/a/4543015/445373

Microsoft. (2001, November). *Introduction to ASP.NET and Web Forms*. Retrieved August 29, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/ms973868.aspx#introwebforms_topic1

Microsoft. (2003, July). *Build XML-Based Images in ASP.NET Using Scalable Vector Graphics*. (D. Forbes, Editor) Retrieved March 2012, from MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/cc164114.aspx

Microsoft. (2010). *Inheritance (C# Programming Guide)*. Retrieved September 23, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/ms173149%28v=vs.100%29.aspx
Microsoft. (2010). *LINQ (Language-Integrated Query)*. Retrieved August 30, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/bb397926%28v=vs.100%29.aspx
Microsoft. (2010). *LINQ to SQL*. Retrieved March 2012, from MSDN: http://msdn.microsoft.com/en-us/library/bb386976%28v=vs.100%29.aspx
Microsoft. (2010). *Using Intellisense*. Retrieved August 29, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/hcw1s69b%28v=vs.100%29.aspx
Microsoft. (2010, March 16). *VML to SVG Migration Guide*. Retrieved August 29, 2012, from Microsoft Download Center: http://www.microsoft.com/en-us/download/details.aspx?id=5224
Microsoft. (2011, September 22). *How to serialize an object to XML by using Visual C#.* Retrieved October 5, 2012, from Microsoft Support: http://support.microsoft.com/kb/815813
Microsoft. (2012, March 13). *ASP.NET MVC 3*. Retrieved March 2012, from MSDN: http://msdn.microsoft.com/en-us/library/gg416514%28v=vs.98%29.aspx
Microsoft. (2012). *Team Foundation Server 2012*. Retrieved August 31, 2012, from MSDN: http://msdn.microsoft.com/en-us/vstudio/ff637362
Microsoft DevLabs. (n.d.). *Code Contracts*. Retrieved March 2012, from MSDN: http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx
Microsoft. (n.d.). *Entity Framework*. Retrieved August 31, 2012, from MSDN: http://msdn.microsoft.com/en-us/data/ef.aspx
Microsoft. (n.d.). *FormCollection Class.* Retrieved September 1, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/system.web.mvc.formcollection%28v=vs.100%29.aspx
Microsoft. (n.d.). *IDisposable Interface*. Retrieved September 10, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/system.idisposable%28v=vs.100%29.aspx
Microsoft. (n.d.). *List<T>.Sort Method (IComparer<T>).* Retrieved September 26, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/234b841s%28v=vs.100%29.aspx
Microsoft. (n.d.). *Microsoft DreamSpark*. Retrieved January 2012, from Microsoft DreamSpark: https://www.dreamspark.com
Microsoft. (n.d.). *MVC: Official Microsoft Site*. Retrieved March 2012, from Microsoft ASP.net: http://www.asp.net/mvc
Microsoft. (n.d.). *Stored Procedure Basics*. Retrieved August 31, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/ms191436%28v=sql.105%29
Microsoft. (n.d.). *ValidationExtensions.ValidationMessageFor<TModel, TProperty> Method (HtmlHelper<TModel>, Expression<Func<TModel, TProperty>>, String)*. Retrieved September 22, 2012, from MSDN: http://msdn.microsoft.com/en-us/library/ee703502%28v=vs.100%29.aspx
Miller, J. (2009, June). *The Unit Of Work Pattern And Persistence Ignorance.* Retrieved September 4, 2012, from MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/dd882510.aspx

Miller, R. (n.d.). *Entity Framework Development Workflows.* Retrieved October 8, 2012, from Data Developer Center: http://msdn.microsoft.com/en-us/data/jj590134

Mischel, J., & chance, c. o. (2010, September 13). *Using a MemoryStream with FileStreamResult possible?* Retrieved September 25, 2012, from StackOverflow: http://stackoverflow.com/questions/3702448/using-a-memorystream-with-filestreamresult-possible

Mitchell, M., Nazim, G., & Muñoz, C. (2008, September 15). *What's the best method in ASP.NET to obtain the current domain?* Retrieved September 26, 2012, from StackOverflow: http://stackoverflow.com/questions/61817/whats-the-best-method-in-asp-net-to-obtain-the-current-domain

MonoDevelop. (2005). *MonoDevelop*. Retrieved March 2012, from MonoDevelop: http://monodevelop.com

Morgan, S. (2008, September 11). *How do you perform a CROSS JOIN with LINQ to SQL?* Retrieved October 4, 2012, from StackOverflow: http://stackoverflow.com/a/56612/445373

Mrnka, L. (2011, March 26). *composite key as foreign key.* Retrieved August 28, 2012, from StackOverflow: http://stackoverflow.com/a/5441569/445373

Mrnka, L. (2011, March 27). *EF 4.1 Code-first vs Model/Database-first.* Retrieved October 8, 2012, from StackOverflow: http://stackoverflow.com/questions/5446316/ef-4-1-code-first-vs-model-database-first

Mrnka, L. (2011, December 18). *EF Code first, how to register same table name with different schema?* Retrieved September 19, 2012, from StackOverflow: http://stackoverflow.com/a/8551961/445373

Mrnka, L. (2011, May 2). *EF Code-First table per type with base class DbSet.* Retrieved October 3, 2012, from StackOverflow: http://stackoverflow.com/a/5862211/445373

Niessner, J. (2010, March 23). *The parameters dictionary contains a null entry for parameter 'productId' of non-nullable type 'System.Int32' for method 'System.Web.Mvc.ViewResult Edit(Int32)' in 'WebUI.Controllers.AdminController'*. Retrieved September 21, 2012, from StackOverflow: http://stackoverflow.com/a/2500046/445373

Nozik, G. (2011, September 18). *how to use views in code first entity framework.* Retrieved September 28, 2012, from StackOverflow: http://stackoverflow.com/a/7461461/445373

Orion, E. (2007, December 20). *Norway mandates government use of ODF.* Retrieved March 2012, from The Inquirer: http://www.theinquirer.net/inquirer/news/1029860/norway-mandates-government-odf

Orsich, A. (2011, January 5). *The conversion of a datetime2 data type to a datetime data type resulted in an out-of-range value.* Retrieved September 29, 2012, from StackOverflow: http://stackoverflow.com/a/4609323/445373

Outercurve Foundation. (2012, August 28). *Home*. Retrieved August 29, 2012, from NuGet Gallery: http://nuget.org/

PCI Security Standards Council, LLC. (2008). PCI Quick Reference Guide. *Understanding the Payment Card Industry Data Security Standard version 1.2*.

Retrieved September 8, 2012, from https://www.pcisecuritystandards.org/documents/pci_ssc_quick_guide.pdf

Perkins, B. (2010). *Create a Random or Sequential GUID using ASP.NET and C#.* Retrieved September 2, 2012, from TheBestCSharpProgrammerInTheWorld.com: http://www.thebestcsharpprogrammerintheworld.com/blogs/Create-a-Random-or-Sequential-GUID.aspx

raduenuca. (2011, March 12). *Re: MVC 3 Connection String without SQLEXPRESS.* Retrieved August 22, 2012, from The Official Microsoft ASP.NET Forums: http://forums.asp.net/post/4339582.aspx

Richardson, T. (2010, April 19). *Timepicker Addon for jQuery UI Datepicker.* Retrieved September 8, 2012, from Trent Richardson | Practical Web Development: http://trentrichardson.com/2010/04/19/timepicker-addon-for-jquery-ui-datepicker/

Russo, M. (2008, January 14). *The NOT IN clause in LINQ to SQL.* Retrieved October 4, 2012, from Microsoft LINQ Books: http://programminglinq.com/blogs/marcorusso/archive/2008/01/14/the-not-in-clause-in-linq-to-sql.aspx

Selenium. (n.d.). *Selenium*. Retrieved March 2012, from Selenium - Web Browser Automation: http://seleniumhq.org

serialhobbyist. (2009, July 14). *Why isn't there an XML-serializable dictionary in .NET?* Retrieved October 5, 2012, from StackOverflow: http://stackoverflow.com/questions/1124597/why-isnt-there-an-xml-serializable-dictionary-in-net

Sheldon, R. (2012, May 24). *SQL Server APPLY Basics.* Retrieved October 4, 2012, from simple-talk: http://www.simple-talk.com/sql/t-sql-programming/sql-server-apply-basics/

Skillsoft. (2011). *About Books24x7*. Retrieved August 29, 2012, from Skillsoft: http://www.skillsoft.com/Books24x7/About_Us/default.asp

Slauma. (2011, December 22). *Problems using TPT (Table Per Type) in EF 4.2 and deletion of parent objects*. Retrieved September 21, 2012, from StackOverflow: http://stackoverflow.com/a/8605313/445373

Slauma. (2011, October 17). *Validation failed for one or more entities. See 'EntityValidationErrors' property for more details.* Retrieved August 28, 2012, from StackOverflow: http://stackoverflow.com/a/7798264/445373

Smith, B. (2008, November 21). *Why are circular references in Visual Studio a bad practice?* Retrieved September 23, 2012, from StackOverflow: http://stackoverflow.com/a/309198/445373

Spech, P. (2008). *Robert Gagne - Learning Theorist*. Retrieved April 2012, from Boise State: http://edtech2.boisestate.edu/spechtp/575/learningtheorist.html

Stannard, A. (2010, December 7). *How-To: Remote Validation in ASP.NET MVC3*. Retrieved September 11, 2012, from Aaronontheweb: http://www.aaronstannard.com/post/2010/12/07/remote-validation-asp-net-mvc3.aspx

STEM Education Coalition. (n.d.). *Mission*. Retrieved August 30, 2012, from STEM Education Coalition: http://www.stemedcoalition.org/about-us/objectives/

Stock, T., & Krome, P. (2009, May 19). *How can I get this ASP.NET MVC SelectList to work?* Retrieved September 7, 2012, from StackOverflow: http://stackoverflow.com/a/877836/445373

Strahl, R. (2008, February 5). *Linq to SQL DataContext Lifetime Management.* Retrieved September 3, 2012, from Rick Strahl's Web Log: http://www.west-wind.com/weblog/posts/2008/Feb/05/Linq-to-SQL-DataContext-Lifetime-Management

thetoolman. (2008, December). *dyndatetime - jQuery DateTime plugin: progressive enhance your chosen text fields.* Retrieved September 8, 2012, from Google Project Hosting: http://code.google.com/p/dyndatetime/

Thomson Reuters. (2012, March 30). *Robo-readers: the new Teachers' Helper in the U.S.* Retrieved March 30, 2012, from Science World Report: http://www.scienceworldreport.com/articles/2368/20120330/robo-readers-new-teachers-helper-u-s.htm

Ugurlu, T. (2011, September 29). *How to Detect Errors of Our ASP.NET MVC Views on Compile Time - Blow up In My Face Theory*. Retrieved September 19, 2012, from TugberkUgurlu.Com: http://www.tugberkugurlu.com/archive/how-to-detect-errors-of-our-asp-net-mvc-views-on-compile-time-blow-up-in-my-face-theory

Vetter, D. R. (2012, February). Grading Systems from a Computer Science Perspective. (J. Felds, Interviewer)

VsOfficeDeveloper. (2008, May 8). *Office 2007 File Format MIME Types for HTTP Content Streaming.* Retrieved September 5, 2012, from VSOfficeDeveloper: Known Problems, Bugs, and Fixes: http://blogs.msdn.com/b/vsofficedeveloper/archive/2008/05/08/office-2007-open-xml-mime-types.aspx

W3C. (1994). Retrieved February 2012, from The W3C CSS Validation Service: http://jigsaw.w3.org/css-validator/

W3C. (1994). Retrieved February 2012, from The W3C Markup Validation Service: http://validator.w3.org

W3C. (1999, December 24). *HTML 4.01 Specification*. Retrieved August 29, 2012, from W3C: http://www.w3.org/TR/html4/

W3C. (2008, November 26). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Retrieved August 29, 2012, from W3C: http://www.w3.org/TR/REC-xml/

W3C. (2011, June 7). *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. Retrieved August 29, 2012, from W3C: http://www.w3.org/TR/CSS2/

Warren, M. (2007, August 2). *Linq Join Error: The type of one of the expressions in the join clause is incorrect. Type inference failed in the call to 'Join.* Retrieved September 28, 2012, from Microsoft Developer Network Forms: http://social.msdn.microsoft.com/Forums/en-US/linqprojectgeneral/thread/bf98ec7a-cb80-4901-8eb2-3aa6636a4fde/

wesdyer. (2007, January 3). *How Linq to Objects Queries Work.* Retrieved October 8, 2012, from Ye Another Language Geek: http://blogs.msdn.com/b/wesdyer/archive/2007/01/03/how-linq-to-objects-queries-work.aspx

Wheeler, P. (2011, February 25). *Does ReadOnly(true) work with Html.EditorForModel?* Retrieved September 27, 2012, from StackOverflow: http://stackoverflow.com/a/5113091/445373

White Urchin Ventures Inc. (2011). *Indent and Pretty-print XML*. Retrieved March 2012, from XML Formatter: http://www.xmlformatter.net/

Widom, J. (2012, February 24). *From 100 Students to 100,000*. Retrieved March 2012, from SIGMOD Blog: http://wp.sigmod.org/?p=165

Williams, R. (2009, December 15). *Mock a database repository using Moq.* Retrieved August 24, 2012, from Williablog.net: http://williablog.net/williablog/post/2009/12/15/Mock-a-database-repository-using-Moq.aspx

Wilson, B. (2009, October 28). *ASP.NET MVC 2 Templates, Part 3: Default Templates*. Retrieved September 13, 2012, from Brad Wilson: http://bradwilson.typepad.com/blog/2009/10/aspnet-mvc-2-templates-part-3-default-templates.html

Yahoo! Inc. (2012). *Best Practices for Speeding Up Your Web Site*. Retrieved September 8, 2012, from Yahoo! Developer Network: http://developer.yahoo.com/performance/rules.html

Zhu, X. (2010, November 3). *Multiple-Step Based Registration via ASP.NET MVC 2 & WF 4.* Retrieved September 28, 2012, from dotnetslackers.com: http://dotnetslackers.com/articles/aspnet/Multiple-Step-Based-Registration-via-ASP-NET-MVC-2-WF-4.aspx

Appendix A

Spreadsheet File Formats

Appendix A: Spreadsheet File Formats

Three different spreadsheet formats were looked into early on to determine how feasible it would be to utilize them in the first iteration of the re-visioned AGLS. The formats were the Microsoft Excel 2007/2010 format, the OpenOffice/LibreOffice Open Document Spreadsheet Format, and Apple's iWork's Numbers format.

Excel 2007/2010 Format

The Microsoft Office 2007 file format is named .xlsx, but is actually a .zip file. Renaming it to zip and extracting it reveals the contents. The list of files and folders that are relevant for the AGLS this term are as follows:

- [Content_Types].xml
- _rels
    - .rels
- docProps
    - app.xml
    - core.xml
- xl
    - calcChain.xml
    - comments#.xml (where # is a number)
    - sharedStrings.xml
    - styles.xml
    - workbook.xml
    - _rels
        - workbook.xml.rels
    - Drawings
        - drawing1.xml
        - vmlDrawing1.vml
    - printerSettings
        - printerSettings1.bin
    - theme
        - theme#.xml (where # is a number)
    - worksheets
        - sheet#.xml (where # is a number)
        - _rels

- sheet#.xml.rels

The vast majority of the files are xml files. There are also vml files; VMLs are scalar vector graphics similar to SVG, but does not have as much support as SVG (Microsoft, 2003). The rels files are also xml files with a different extension. Only the printerSettings.bin file seems to be in binary.

The content types file explains the extensions that are used, with override tags inside of there explaining any exceptions to the above. Any time there is a _rels folder, this indicates the relationships to different sheets. The sheets are not referred to by name, but by an ID. Every folder contains a _rels subfolder explaining some of the relationships.

The docProps folder is primarily filled with metadata. The app.xml file mentions the person who created the file, and core.xml covers when the sheet was modified.

The xl folder is where most of the data is contained. Formulas are stored in calcChain.xml, and comments are located in comments#.xml, where each # represents the sheet number in question. All strings are placed in sharedStrings.xml for attempted memory reduction, and styles.xml covers formatting. Workbook.xml points to the sheets, and also contains a file version. If there are any drawings, they are placed in a Drawings folder with the numbered drawing files as appropriate.

It is the worksheets folder within the xl folder where the sheets themselves are located. Each sheet contains dimensions, the last cell you were on, and rows of data. While there can be a cols tag to indicate non-standard cell widths, data is parsed row by row in Excel. Each cell can either be a number, an index to the shared strings file, or a formula result. If it is a formula result, then an extra attribute is used to state what the formula is.

OpenDocument Spreadsheet Format

The OpenDocument Spreadsheet format also uses a file and folder .zip structure. When unpacked, it looks like the following:

- content.xml
- meta.xml
- mimetype
- settings.xml
- styles.xml
- Configurations2
    - accelerator
        - current.xml
    - floater
        - *nothing*
    - images
        - Bitmaps
            - *any images go here*
    - menubar
        - *nothing*
    - popupmenu
        - *nothing*
    - progressbar
        - *nothing*
    - statusbar
        - *nothing*
    - toolbar
        - *nothing*
    - toolpanel
        - *nothing*
- META-INF
    - manifest.xml
- Thumbnails
    - thumbnail.png

Similar to the Excel 2007 format, many of the files here are XML based, with few exceptions. For most OpenDocument Spreadsheet files, there are fewer XML files to

worry about. Unlike the Excel XML, OpenDocument's XML is stripped of whitespace at the start. An XML beautifier was used to make it readable for the human eye.

The meta.xml file has some data about when the file was created, and what program made it. The mimetype file is a simple text file containing the mimetype of the ODS format: in this case, it is application/vnd.oasis.opendocument.spreadsheet that is inside. The settings.xml file controls various settings and flags regarding the individual file in question, such as what sheet the user was on last. The styles.xml file defines default styles for numbers, currency, and text, while also including others.

The content.xml file contains the majority of the content. Limited styles and font faces are placed here, and the sheets are in the office body. Each sheet contains some column data for style formatting, while each the contents of each cell are handled row by row. All "empty" rows are assumed to have at least one empty cell. Any cells next to each other with the same content have an attribute indicating how many columns have the consecutive duplicate information, including the empty cell. The number of repeats is not infinite: the upper bound is the number of columns actually in use. There is no shared strings file: strings are placed as is. Formulas are stored as cell attributes, with a strange convention of preceding all cells with a period and surrounding them in [braces].

Numbers (iWork) Format

The .numbers file is technically a directory. Using a program such as 7-Zip File Manager revealed the index.xml file inside. As the XML itself was compressed down to one line, an XML formatter was needed to make the XML readable for studying purposes (White Urchin Ventures Inc, 2011).

It should be noted that many of these tags contain IDs referring to Objective C data types such as NSArray and NSMutableArray. With that knowledge out of the way:

the first few tags inside the core document deal with the various styles available for the sheet. The actual workspaces follow. Each workspace is composed of more than just a single sheet: in Numbers, each sheet can be placed in an arbitrary position on the canvas. To get to the actual sheet data, one would have to travel through these tags: ls:workspace, ls:page-info, ls:layers, sf:layer, sf:drawables, sf:tabular-info. From there, diving into the sf:tabular-model tag and its children will eventually lead you to the table data.

It is perhaps ironic that while Apple prides itself as being a company for great technology that is easy to use, the underlying information is rather convoluted. Focusing on just the file size alone for a moment, I had a spreadsheet I did some work on roughly half a year. Analyzing just the XML part, I had a file size of 3.75 MB with as little whitespace as possible. After beautifying the XML and saving it, the size ballooned to 8.92 MB.

Appendix B

Entity Framework: What's First?

Appendix B: Entity Framework: What's First?

The Entity Framework is an object relational framework that allows mapping database tables to plain old code objects. What makes Entity Framework unique is that there are three ways (four ways in some circles) to allow mapping the tables to objects and vice~versa. The following paragraphs will help explain these different ways. The data below has been mainly written or explained by Rowan Miller and Ladislav Mrnka for future reference. (Mrnka, EF 4.1 Code-first vs Model/Database-first, 2011) (Miller R.)

Code First

The code first way of writing database objects means that the programmer creates all of the objects, and optionally sets up mappings and restrictions using either attributes or the fluent API. The programmer has full control over how the database is designed and what data is supplied to the database when the application loads. However, such control over the code means that the database cannot be fully utilized. Any manual changes that could be performed such as adding indexes could potentially be lost should the code determine that the database has to be dropped and re-created. Likewise, views cannot be created in code first, especially if it's a new database: it is implied that stored procedures would have the same problem (Nozik, 2011).

Database First

The database first way of writing code usually implies that a database already exists to work with. Entity Framework is able to analyze the tables, views, and stored procedures of a database and map them to coded objects for the author. Changes to both the database and the code can be made if planned carefully. Any updates to the database will be reflected by Entity Framework. If any extra methods, properties, indexers, etc.

need to be added to the objects for custom behavior, partial classes can be used to extend the created objects without losing the database generated objects. However, it is not clear if there is a seeding method available with the database first method.

Model First

The model first method allows people who do not like to think about coding or databases, but needs to create a database anyway. The primary tool in this case is a GUI to draw the model out. Entity Framework can then generate the core objects based on the drawn model. Similar to database first, partial classes can be used to extend the generated objects. However, it also shares the code first flaws of not being able to update the database with indexes easily.

Appendix C

Regressions from the Original AGLS

Appendix C: Regressions from the Original AGLS

While the concept of the original AGLS was preserved when building the newer AGLS, not all of the features were duplicated or re-created. His section seeks to explain what was left out.

As this AGLS's focus was strictly on Excel, none of the Microsoft Access features were brought in. Likewise, the grade book functionality is also missing. In its place is the ability to export grades to any common file format to then bring into the grade book of their choice. Exporting to specific grade book software was not considered to be in scope.

Both versions of the AGLS required sheets to be named properly. The original version also allowed for alternate names of sheets to be utilized. The current version only allowed for the same name.

Both versions of the AGLS could grade formulas and cell values. The original version had support for ensuring that a cell value was a number, whereas the newer one was limited to integers. The older AGLS included support for numbers to be given a minimum target, a maximum target, or a range.

The original version of the AGLS included support for grading a cell's formatting. This was an original, if unspecified, target for the newer AGLS, but time constraints prevented its completion.

Going by the code, both versions of the AGLS had some basic support for scenario manager. This statement is strange considering the interview data gathered (see Table 2 for who was interviewed): a number of professors wanted scenario manager support. Either way, both versions looked into similar items: namely, the names of the

scenarios, the cells affected, and their values. The original version wanted to support some form of running the scenario manager, whereas the newer version looked into supporting if a scenario was locked by a user or hidden from other users.

Basic support of charts was found in the original AGLS code. The newer AGLS does not have support for reading any chart data. If that needs to be implemented in the future, the EPPlus package has native support for chart reading.

The original AGLS code also includes support for page orientation and limited solver support. Page orientation has some support in EPPlus, but solver support would have to be coded manually similar to how the scenario manager items were handled.

Appendix D

From Development to Production

Appendix D: From Development to Production

A number of changes to both the code and the configuration files are needed in order to make this production ready. This list also includes items that were never mentioned in any part of scope that would be useful regardless.

Establish a Team

The AGLS in its current iteration was a solo endeavor. More work can be accomplished on this web software if a team of people can be found to work on it, each with different strengths. This application is starting to become too big for one person to maintain alone.

Buy Server Space

This system presently exists on just a development machine. While the source code is under version control, there is no production environment at this point. It will take some time, and likely money, to identify the best place to host this application for a production environment.

Identify Legal/Privacy Issues

In order to utilize a production environment, the website and server must be subject to various laws and regulations regarding the privacy of the users and their data. Sensitive information is stored on in the database such as a first name: if this is not protected, then the AGLS cannot do its job. At this time, it is not known what changes have to be made to the website or the database in order to comply with the needed regulations.

Mail Setup

The Postal plugin by itself is fine: it is merely a library that sends email. The configuration of the email sending, however, is what has to change. In the web.config file, there is an XML tag dictating how email is set up. For development purposes, emails were always being "sent" to a specified hard drive directory, and never to anyone's email inbox. This has to change to be a proper mail server of sorts.

Implement SSH Protocol

When users log in to the AGLS right now, the protocol does not change. It defaults to HTTP. While the passwords are hashed and salted on the database level and the password attribute is used for HTML input elements, it is still not the most secure way of handling this situation. Granted, it is possible to create a self-signed certificate for the purposes of SSH (Lister, 2009). However, it would probably be more legitimate if a certificate was bought from an official company.

Buying and Paying with Cards

At this time, there is a way to "buy" items from the store, and then "pay" for them. First of all, no money is being charged. Second, the form itself is insufficient for a traditional website: more personal information is required for credit card/debit card validation. Third, it would perhaps be better to use a third party service such as Dwolla, Google Checkout, or PayPal to handle the actual purchasing. Fourth, no taxes are calculated for this service right now.

This does not even go into whether card numbers or other identifying information should be stored on the AGLS's database. Granted, there is some information about a person such as their name that is stored. However, credit card numbers have their own rules as far as what to store, how to store them, and for how long, among other rules (PCI

Security Standards Council, LLC, 2008). It may be better to find a third party to handle the credit card data.

Change the Layout/CSS

It is no secret that the vast majority of the styling used is heavily borrowed from the default files provided by creating a non-empty ASP.NET MVC application. While some parts of the initial styling are perhaps useful, a new layout would be more beneficial in the long run.

Different Prices for Various Reasons

For the purposes of this application, buying a class only costs ten United States Dollars. Admittedly, no one actually paid during this project, but that is beside the point. At this point here is no way to change the price for individual colleges or professors for any reason. More specifically, that price is right now hardcoded. If any event were to take place that would cause a price change, it would affect everyone, not just specific users.

Consider Switching to Entity Framework Database First

When the code is brought to a production server, the database will have to be configured, and certain data needs to be filled in right away such as country data. While this is presently done with the code first methodology (see Appendix B for more information), there is a class that, when invoked, drops and re-creates the database. Such a class has no place in the production environment.

Furthermore, by moving to the database first methodology, full control over the database is allowed. Custom indexes and unique keys can be made without having to struggle to code it. Custom properties and indexers can still be used thanks to the partial class support. Lastly, if the database has to change structure, the data will only be lost due

to carelessness of the database administrator. Code first was helpful for the building and fleshing out of the project, but a change may be required for when this is properly hosted.

Reduce Reliance on Fallbacks

A date time picker was chosen for this project due to it being easy to use and having a compatible licensing scheme. However, it worked best when a different jQuery UI theme was used. The theme used for this project was loaded via a CDN. As CDNs can go down, a fallback mechanism would be useful to ensure that the page still works as intended (James, 2012).

Remove Registration Confirmation Shortcuts

When one registers with the AGLS in the development server, the website loads a page that says to check one's email. However, at the bottom of that page is a quick link to jump straight to the confirmation page. Well…if this was a production system, that is where the link would take the user. Instead, the link causes the server to simply activate the user's account and log him or her in right away. This is definitely a bad practice as not only is there no backup confirmation, but the link does not behave like it should. Touching the database for anything other than reading is supposed to be done for POST requests; this was technically a GET request (Korpela, 2003).

Expand Membership Provider

The membership provider that is active in the AGLS is very limiting. It supports being able to identify users and duplicate records. However, there are more features available to the provider such as security questions, password attempts, and more that should be taken advantage of in a production environment.

Use Universal Time for Everything

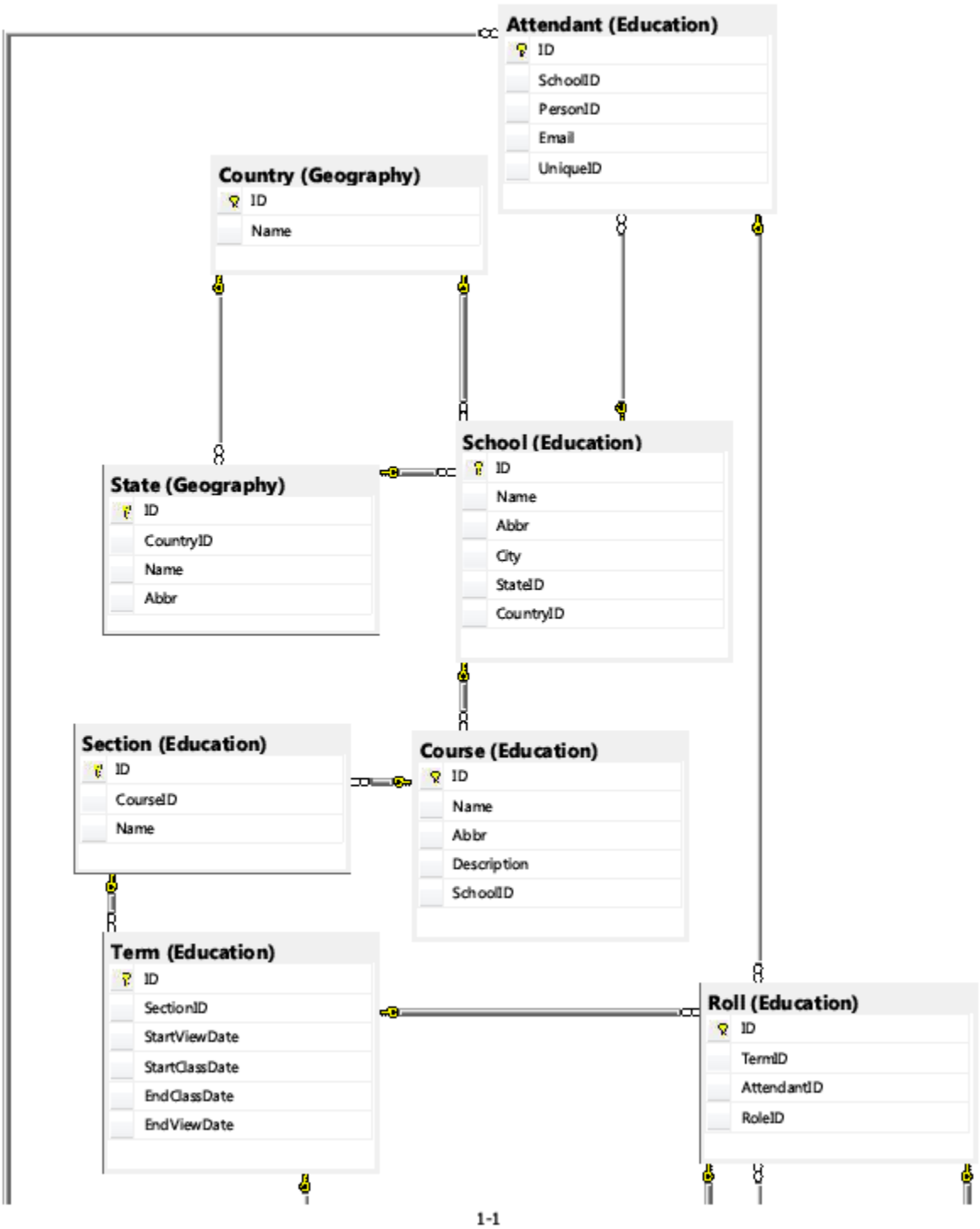
One of the goals of the AGLS for future development is to allow colleges from all over the country, if not the world, to take advantage of this web site and service. At this time, all of the DateTime calls that are taking place simply call the Now property to set the time based on the server's time zone. During development this was always the Eastern Time Zone of the United States. To be consistent and fair across all countries, the UtcNow property should be used instead.

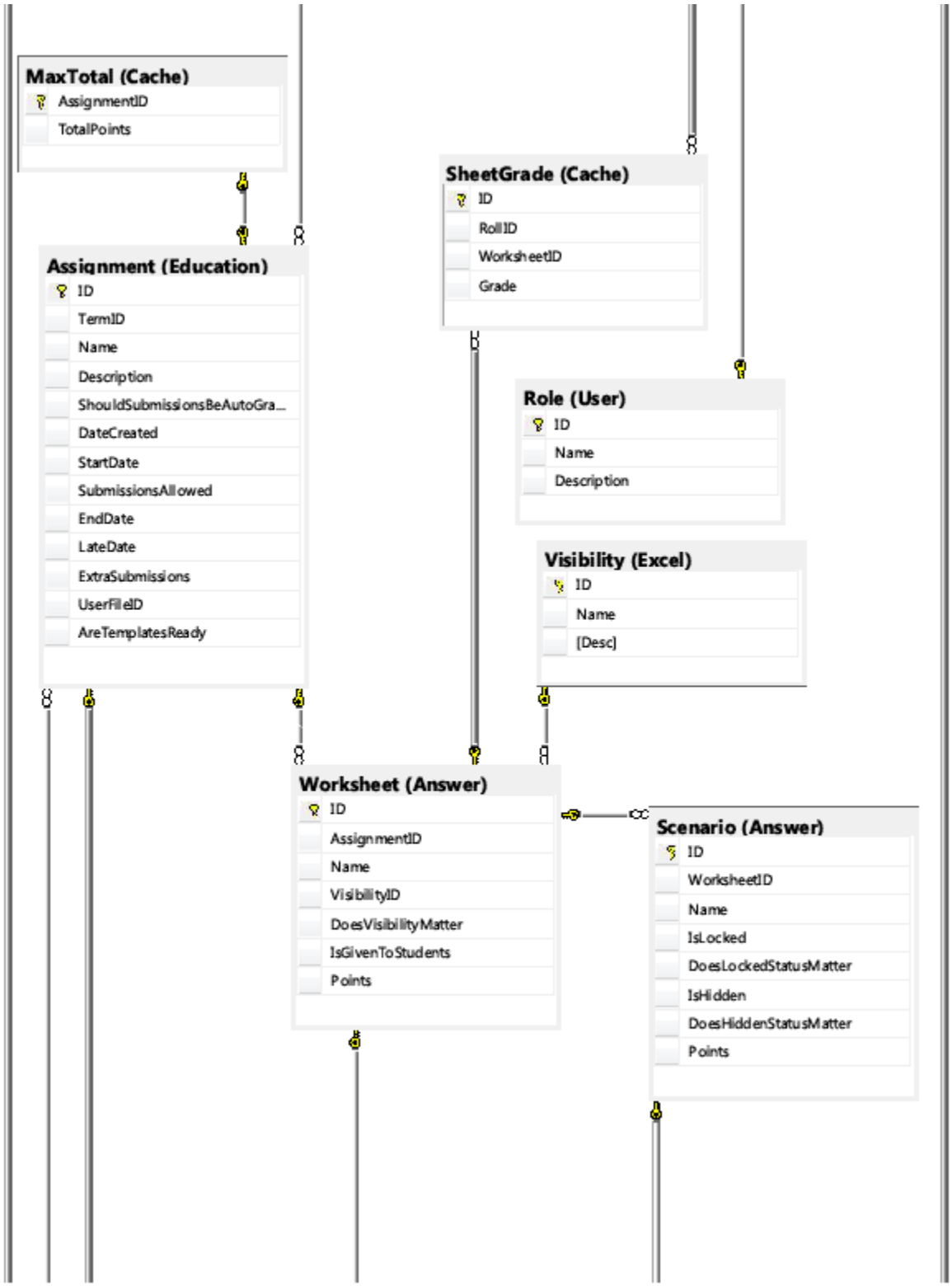
Such a change would require some more modifications to the database structure as it currently stands. There would need to be a time zone table, likely in the Geography schema, which contains the list of time zones. The Person table would also need a foreign key to said table to allow for setting the time zone properly. Each assignment that is created would need to either be set to use universal time from the start, or factor in the time zone of the logged in user.

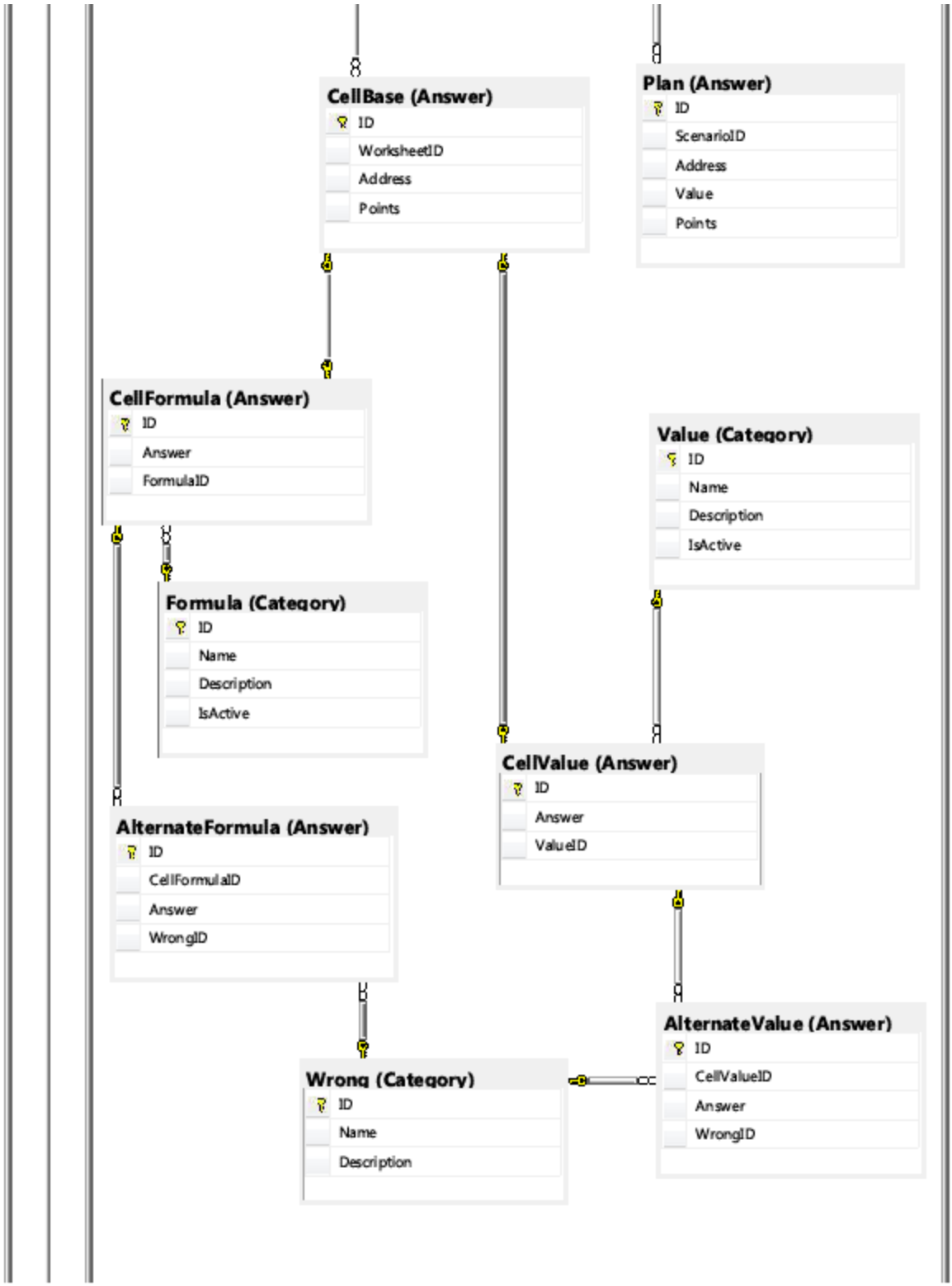
Add More Flexibility to Users

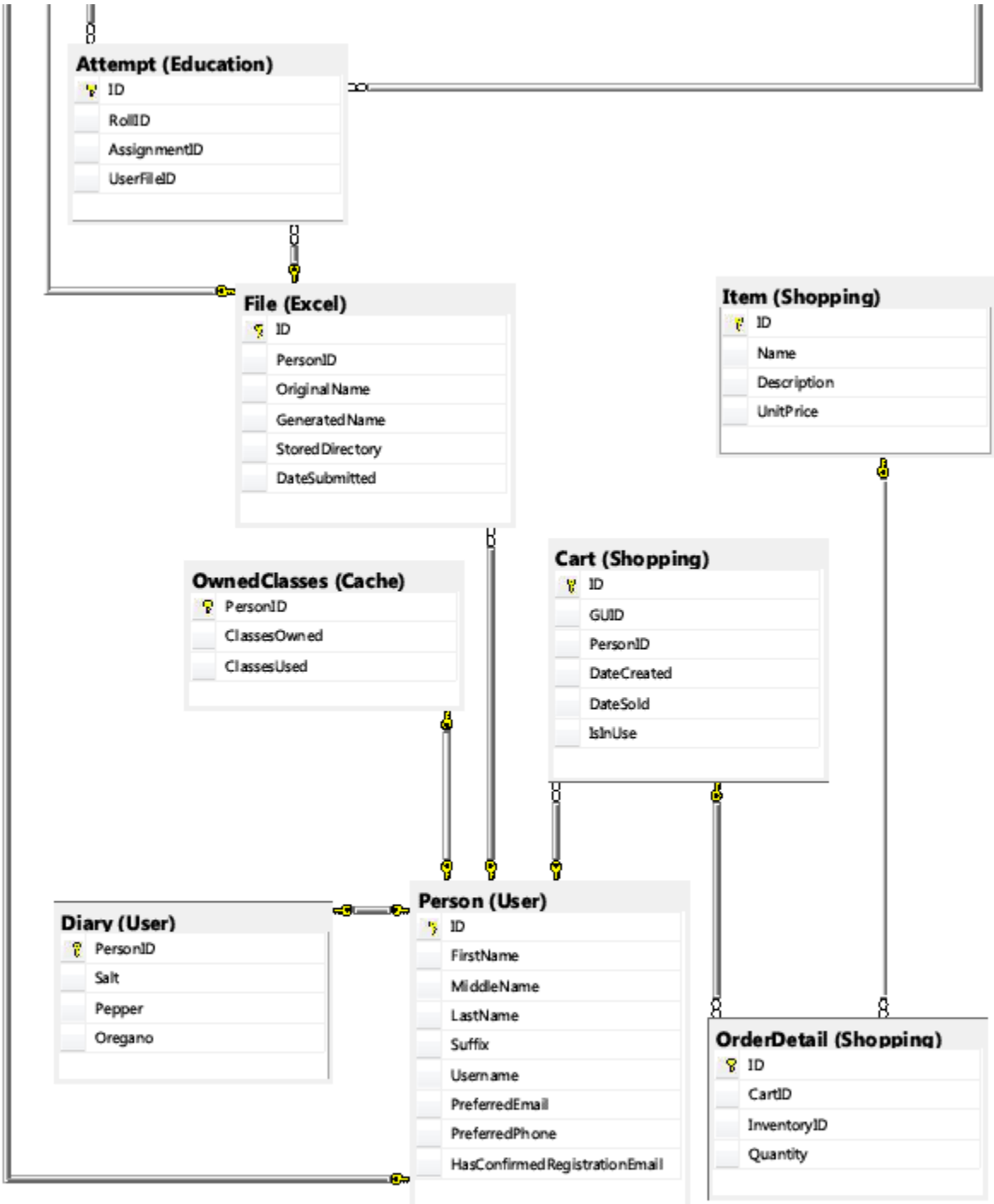
For many of the options in the AGLS, once an item is set, it is locked in forever. There should be more options to allow modifying data in case the original input was erroneous. Such examples would be modifying assignment due dates.

Picture 1: Database Diagram of AGLS

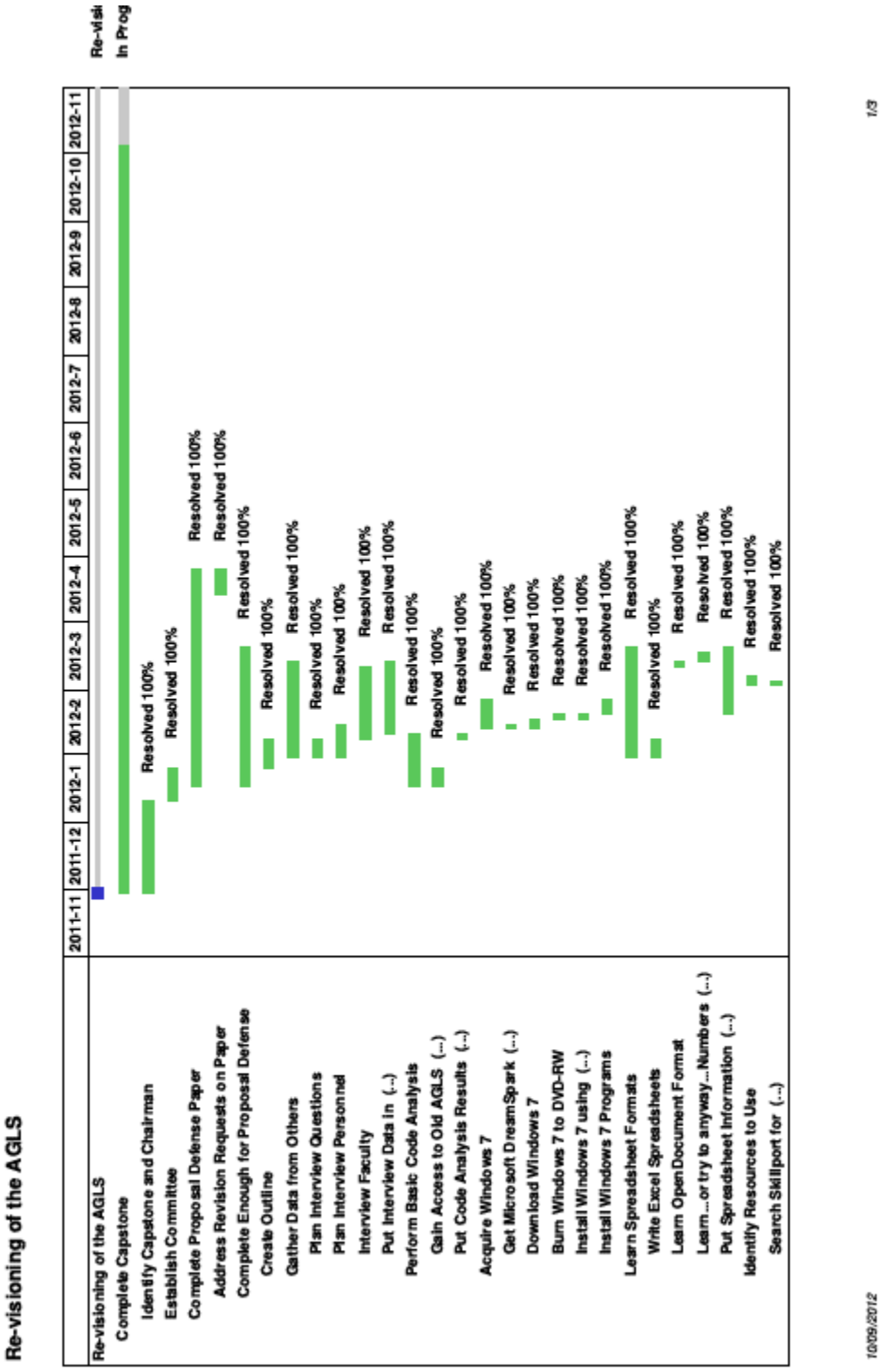


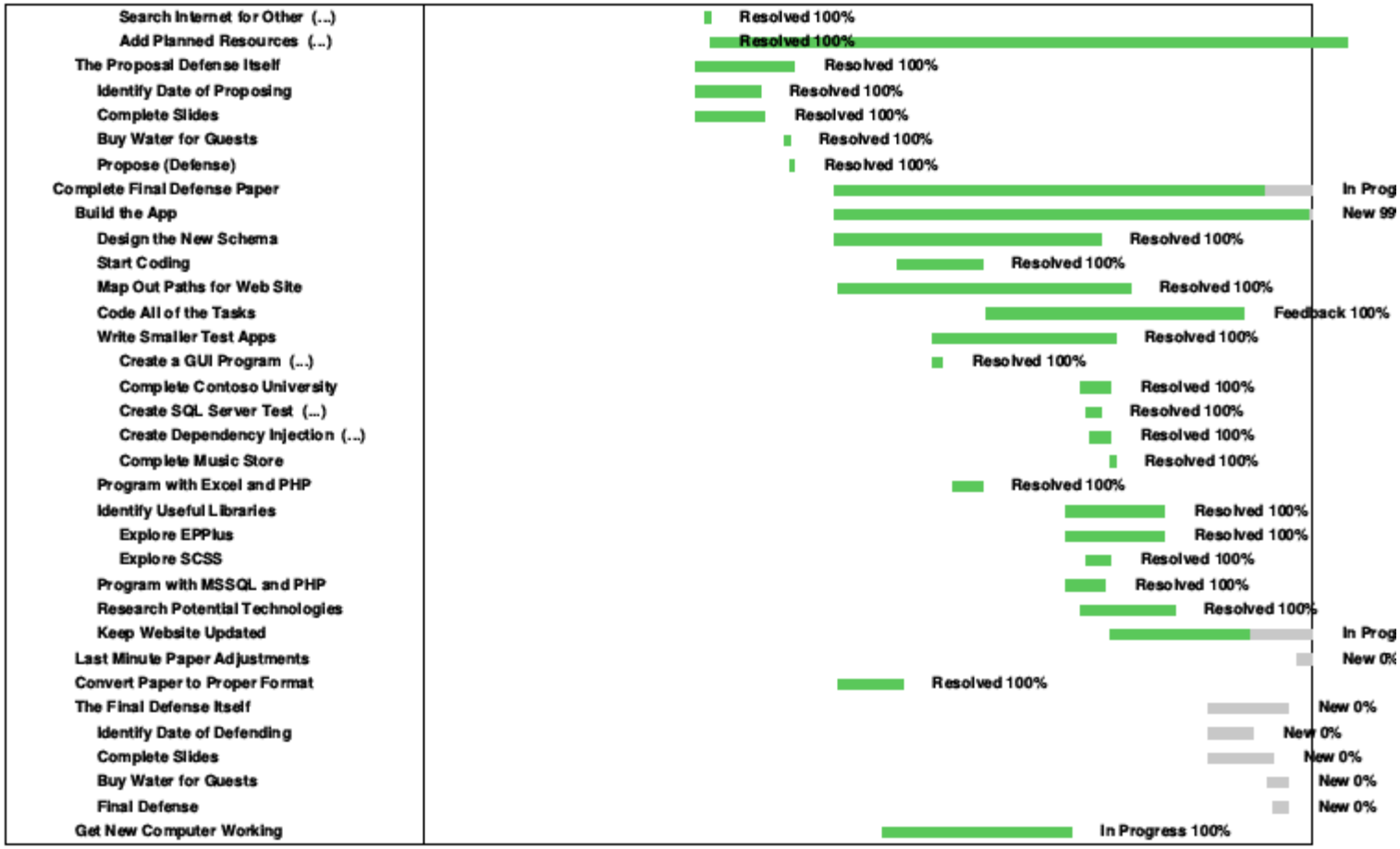

MaxTotal (Cache)
AssignmentID
TotalPoints
SheetGrade (Cache)
ID
RollID
WorksheetID
Grade
Assignment (Education)
ID
TermID
Name
Description
ShouldSubmissionsBeAutoGra...
DateCreated
StartDate
SubmissionsAllowed
EndDate
LateDate
ExtraSubmissions
UserFileID
AreTemplatesReady
Role (User)
ID
Name
Description
Visibility (Excel)
ID
Name
[Desc]
Worksheet (Answer)
ID
AssignmentID
Name
VisibilityID
DoesVisibilityMatter
IsGivenToStudents
Points
Scenario (Answer)
ID
WorksheetID
Name
IsLocked
DoesLockedStatusMatter
IsHidden
DoesHiddenStatusMatter
Points


CellBase (Answer)
ID
WorksheetID
Address
Points
Plan (Answer)
ID
ScenarioID
Address
Value
Points
CellFormula (Answer)
ID
Answer
FormulaID
Value (Category)
ID
Name
Description
IsActive
Formula (Category)
ID
Name
Description
IsActive
CellValue (Answer)
ID
Answer
ValueID
AlternateFormula (Answer)
ID
CellFormulaID
Answer
WrongID
AlternateValue (Answer)
ID
CellValueID
Answer
WrongID
Wrong (Category)
ID
Name
Description


Attempt (Education)
ID
RollID
AssignmentID
UserFileID
File (Excel)
ID
PersonID
OriginalName
GeneratedName
StoredDirectory
DateSubmitted
Item (Shopping)
ID
Name
Description
UnitPrice
OwnedClasses (Cache)
PersonID
ClassesOwned
ClassesUsed
Cart (Shopping)
ID
GUID
PersonID
DateCreated
DateSold
IsInUse
Diary (User)
PersonID
Salt
Pepper
Oregano
Person (User)
ID
FirstName
MiddleName
LastName
Suffix
Username
PreferredEmail
PreferredPhone
HasConfirmedRegistrationEmail
OrderDetail (Shopping)
ID
CartID
InventoryID
Quantity

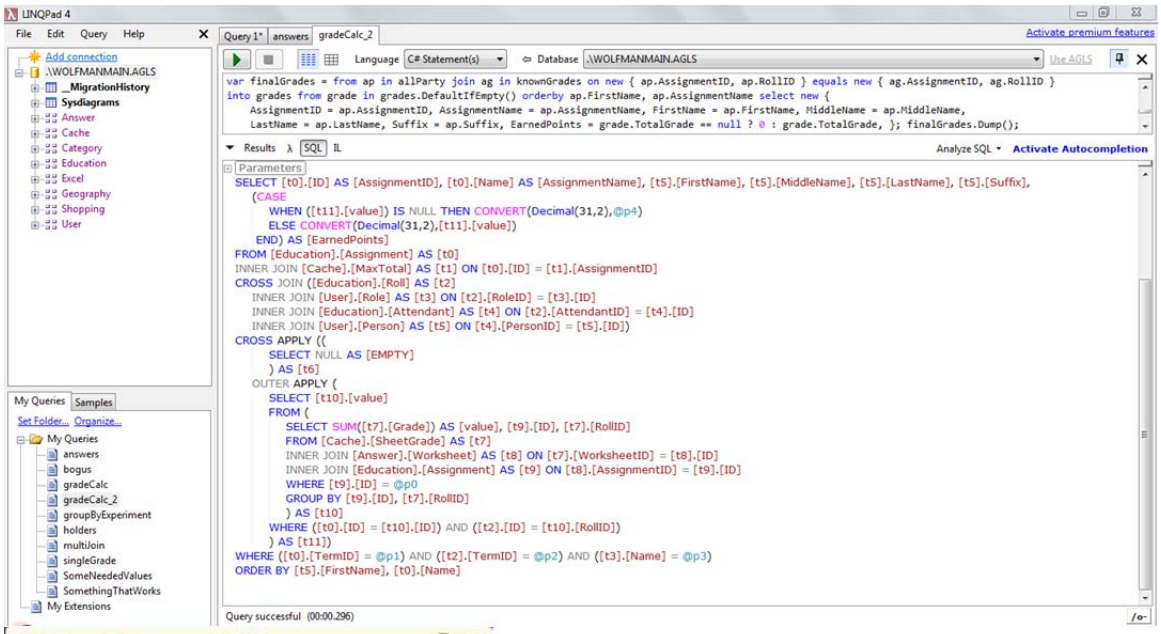
Picture 2: Time Log of Progress



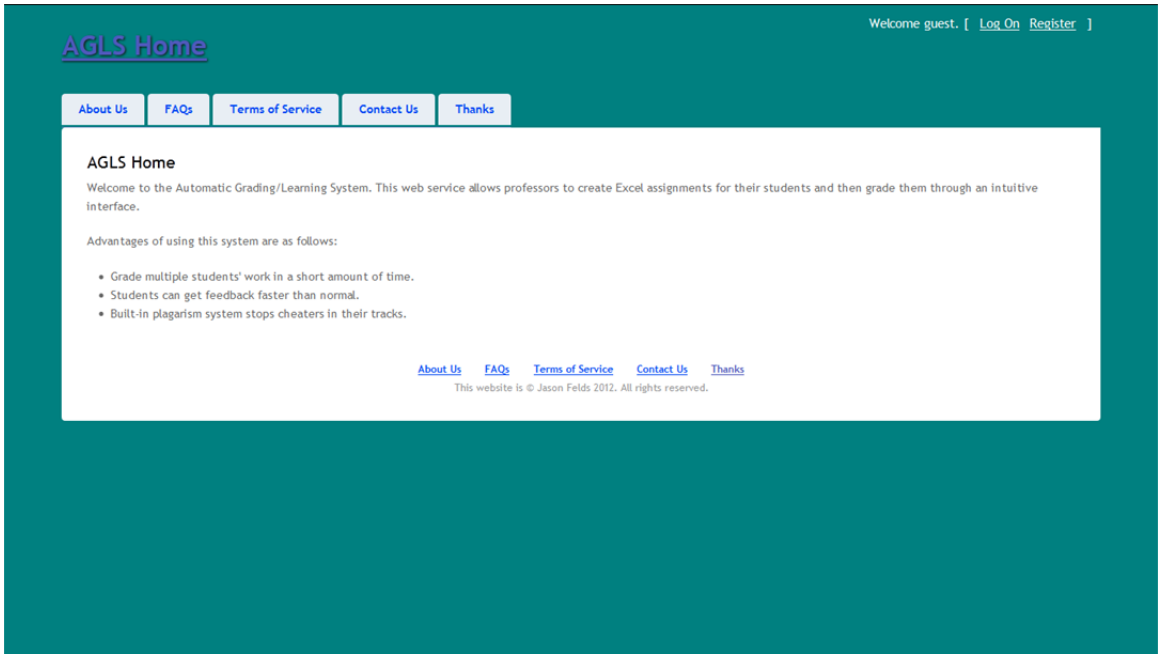

Search Internet for Other (...)
Resolved 100%
Add Planned Resources (...)
Resolved 100%
The Proposal Defense Itself
Resolved 100%
Identify Date of Proposing
Resolved 100%
Complete Slides
Resolved 100%
Buy Water for Guests
Resolved 100%
Propose (Defense)
Resolved 100%
Complete Final Defense Paper
In Prog
Build the App
New 99
Design the New Schema
Resolved 100%
Start Coding
Resolved 100%
Map Out Paths for Web Site
Resolved 100%
Code All of the Tasks
Feedback 100%
Write Smaller Test Apps
Resolved 100%
Create a GUI Program (...)
Resolved 100%
Complete Contoso University
Resolved 100%
Create SQL Server Test (...)
Resolved 100%
Create Dependency Injection (...)
Resolved 100%
Complete Music Store
Resolved 100%
Program with Excel and PHP
Resolved 100%
Identify Useful Libraries
Resolved 100%
Explore EPPlus
Resolved 100%
Explore SCSS
Resolved 100%
Program with MSSQL and PHP
Resolved 100%
Research Potential Technologies
Resolved 100%
Keep Website Updated
In Prog
Last Minute Paper Adjustments
New 0%
Convert Paper to Proper Format
Resolved 100%
The Final Defense Itself
New 0%
Identify Date of Defending
New 0%
Complete Slides
New 0%
Buy Water for Guests
New 0%
Final Defense
New 0%
Get New Computer Working
In Progress 100%

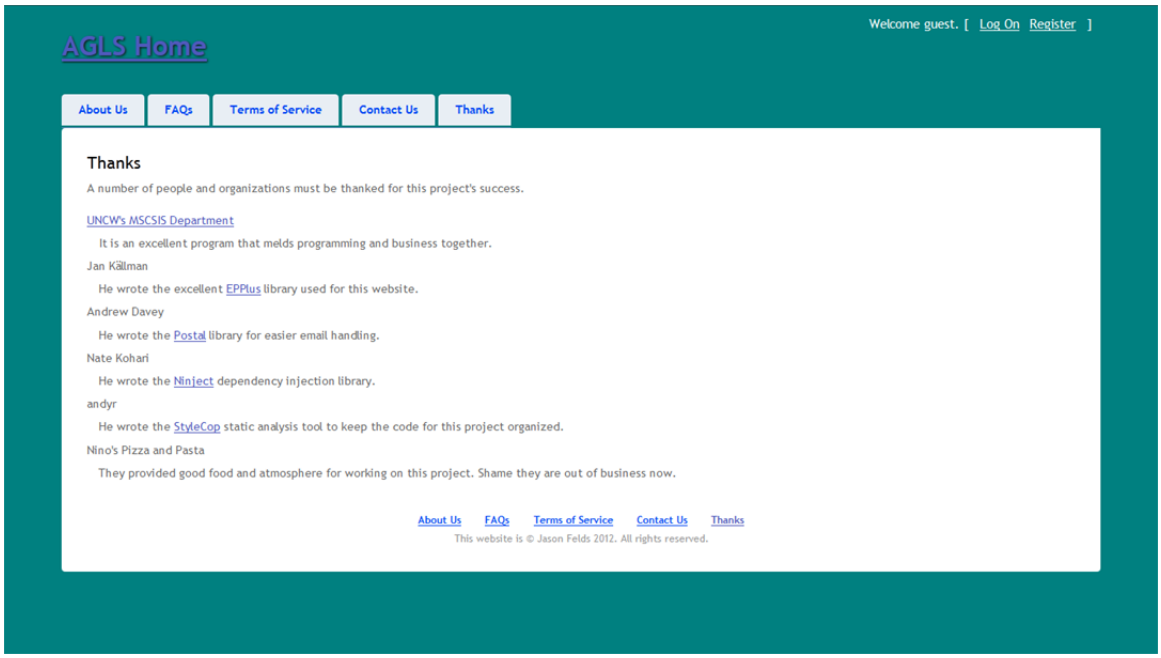

Find and Buy the new machine
Download Needed Software
Download VS2010
Download VS2010 (...)
Download SQL Server
Meet with Acting Chair
Study the new MIS 213 book
Update the Paper
Establish Online Project Planner
Look Into All Options
Buy & Configure VPS Space
Set Up Redmine and DNS
Resolved 100%
Resolved 100%
Resolved 100%
Resolved 100%
Resolved 100%
In Progress
Resolved 100%
In Progress 90%
Resolved 100%
Resolved 100%
Resolved 100%
Resolved 100%

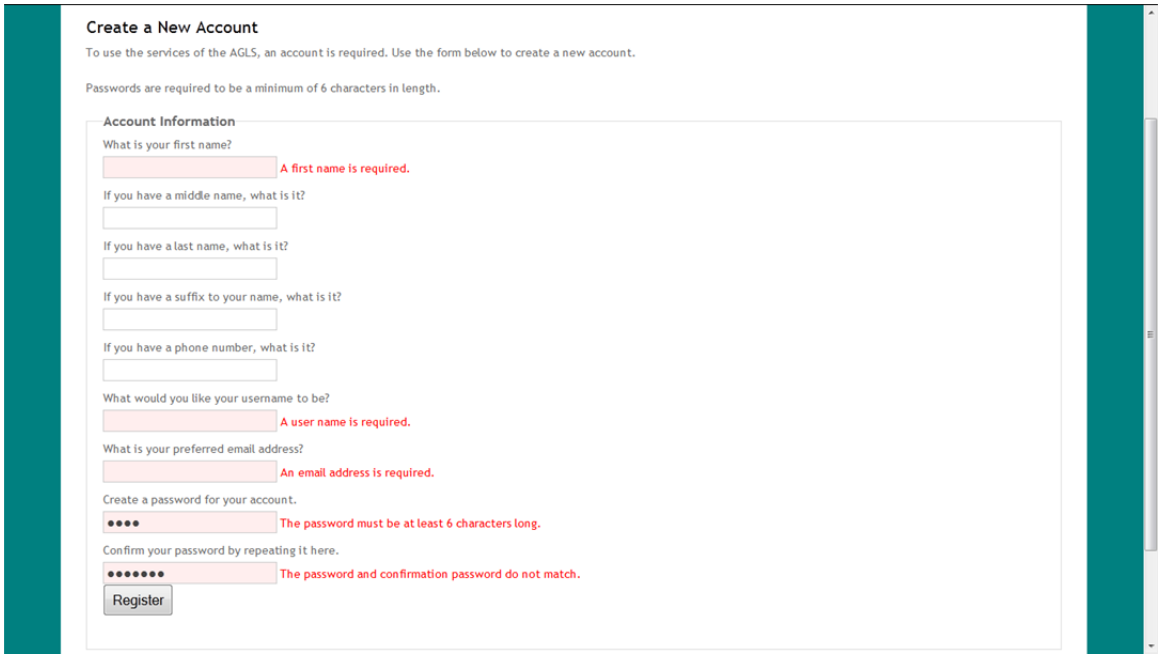
Picture 3: Assorted Development Pictures

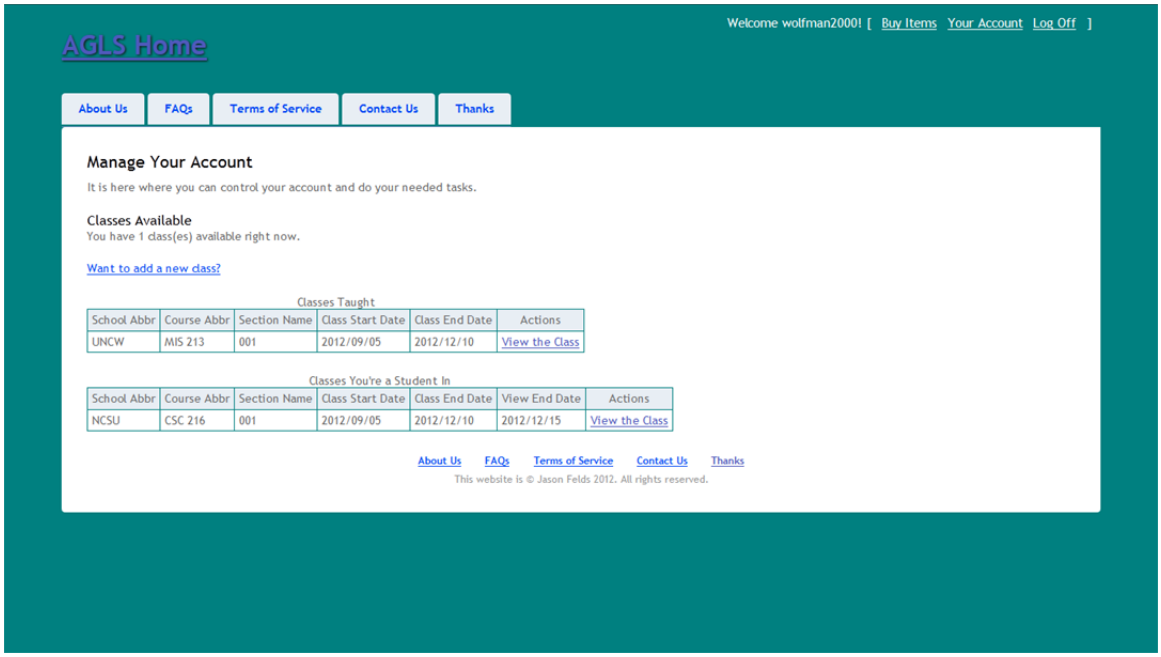


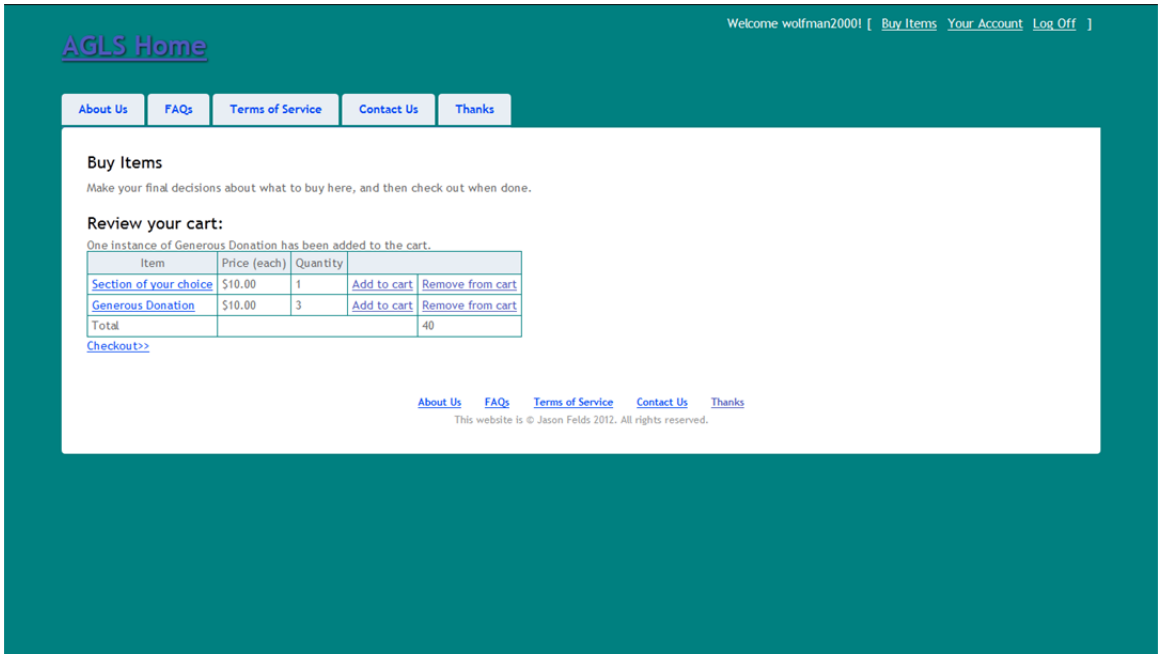
Picture 4: Assorted Website Pictures






Create a New Account
To use the services of the AGLS, an account is required. Use the form below to create a new account.
Passwords are required to be a minimum of 6 characters in length.
Account Information
What is your first name?
A first name is required.
If you have a middle name, what is it?
If you have a last name, what is it?
If you have a suffix to your name, what is it?
If you have a phone number, what is it?
What would you like your username to be?
A user name is required.
What is your preferred email address?
An email address is required.
Create a password for your account.
The password must be at least 6 characters long.
Confirm your password by repeating it here.
The password and confirmation password do not match.
Register


Welcome wolfman2000! [Buy Items Your Account Log Off]
AGLS Home
About Us
FAQs
Terms of Service
Contact Us
Thanks
Manage Your Account
It is here where you can control your account and do your needed tasks.
Classes Available
You have 1 class(es) available right now.
Want to add a new class?
Classes Taught
School Abbr | Course Abbr | Section Name | Class Start Date | Class End Date | Actions
UNCW | MIS 213 | 001 | 2012/09/05 | 2012/12/10 | View the Class
Classes You're a Student In
School Abbr | Course Abbr | Section Name | Class Start Date | Class End Date | View End Date | Actions
NCSU | CSC 216 | 001 | 2012/09/05 | 2012/12/10 | 2012/12/15 | View the Class
About Us FAQs Terms of Service Contact Us Thanks
This website is © Jason Felds 2012. All rights reserved.


Welcome wolfman2000! [Buy Items Your Account Log Off]
AGLS Home
About Us
FAQs
Terms of Service
Contact Us
Thanks
Buy Items
Make your final decisions about what to buy here, and then check out when done.
Review your cart:
One instance of Generous Donation has been added to the cart.
Item
Price (each)
Quantity
Section of your choice
$10.00
1
Add to cart
Remove from cart
Generous Donation
$10.00
3
Add to cart
Remove from cart
Total
40
Checkout>>
About Us FAQs Terms of Service Contact Us Thanks
This website is © Jason Felds 2012. All rights reserved.

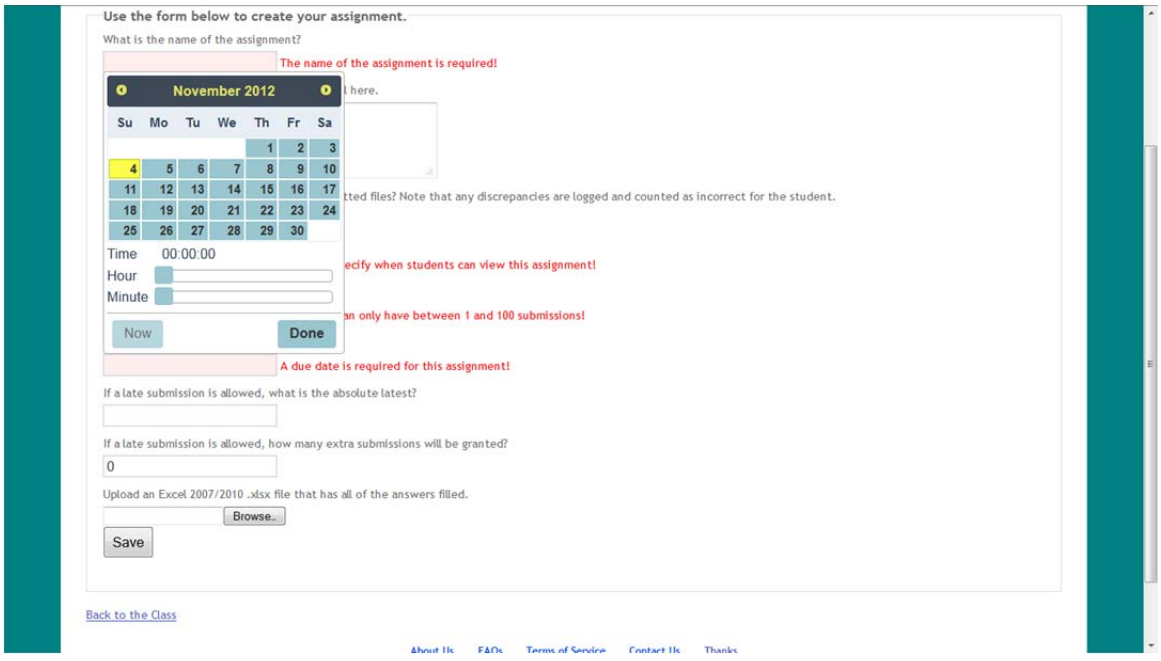

Use the form below to create your assignment.
What is the name of the assignment?
The name of the assignment is required!
November 2012
Su Mo Tu We Th Fr Sa
Time 00:00:00
Hour
Minute
Now
Done
A due date is required for this assignment!
If a late submission is allowed, what is the absolute latest?
If a late submission is allowed, how many extra submissions will be granted?
0
Upload an Excel 2007/2010 .xlsx file that has all of the answers filled.
Browse...
Save
Back to the Class

Table 1: Web Forms vs. MVC

The following resources were used to build this table:

- http://blog.mikecouturier.com/2011/03/aspnet-web-forms-versus-net-mvc.html
- http://eric.polerecky.com/a-completely-biased-comparison-of-aspnet-mvc-and-webforms/
- http://www.slideshare.net/crederajfischer/aspnet-mvc-vs-web-forms

ASP.NET Web Forms		**ASP.NET MVC**	
Pros	**Cons**	**Pros**	**Cons**
Wide spread	Similar to making a desktop application	Works with HTTP protocol.	Usually takes longer to develop than Web Forms.
Similar to making a desktop application	View State does not work well with HTTP protocol.	Uses established Model View Controller approach to isolate the different parts.	No built-in form controls.
Quick to Deploy (Rapid Application Development, or RAD)	HTML Validation harder to pull off.	Convention over configuration: built-ins work, but you have the power to adjust anything as needed.	Requires more of an understanding of web development topics and general programming.
Integration Testing possible (Selenium)	Not all Web Forms Controls map to one HTML element.	Cleaner URLs possible out of the gate: better for SEO purposes.	Supports less third party libraries.
Supports more third party libraries.	Content and Layout are tightly integrated, not loosely coupled.	Easier to validate HTML.	No View State means the user must control form repopulation.
	Harder to unit test Code Behind pages.	Integration Testing possible (Selenium)	
	Harder to maintain code.	Unit Testing made easier.	
	SEO is harder to pull off.	No Code Behind pages.	
		Relatively simple to add or extend features.	
		Easier to maintain code.	

Table 2: Interview Data

Person	Department	Relevant Class[es]
Marni Ferner	Computer Science	CSC 105
Judith Gebauer	Info Systems and Operations Mgmt.	MIS 213
Thomas Janicki	Info Systems and Operations Mgmt.	MIS 213
Douglas Kline	Info Systems and Operations Mgmt.	MIS 213, MIS 413
Lorraine Lee	Accountancy and Business Law	MSA 522, ACG 445
Laurie Patterson	Computer Science	CSC 105
Brian Reinicke	Info Systems and Operations Mgmt.	MIS 213
Rebecca Sawyer	Accountancy and Business Law	ACT 306
Ron Vetter	Computer Science	CSC 105

A number of professors were interviewed during the early planning process. These interviews were used to gather data on how to best pursue this project. They are listed in alphabetical order, with the last name coming first.

Table 3: Skillport Books Utilized

Focus	Title	Publisher
C#	Beginning Visual C# 2010	Wrox Press
C#	Pro LINQ: Language Integrated Query in C# 2010	Apress
C#	Pro C# 2010 and the .NET 4 Platform, Fifth Edition	Apress
C#	Professional C# 4 and .NET 4	Wrox Press
VB.NET	Pro VB 2010 and the .NET 4.0 Platform	Apress
VB.NET	Professional Visual Basic 2010 and .NET 4	Wrox Press
ASP.NET MVC	Pro ASP.NET MVC 3 Framework, Third Edition	Apress
ASP.NET MVC	Professional ASP.NET MVC 3	Wrox Press